Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Iordan Pentchev

Robotersteuerung in interaktiver Testumgebung:
Optimierte Anfahrt an extrapolierten Aufschlagpunkt
eines geworfenen Objekts

Fakultät Technik und Informatik
Department Informations- und
Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and
Electrical Engineering

Iordan Pentchev

Robotersteuerung in interaktiver Testumgebung: Optimierte Anfahrt an extrapolierten Aufschlagpunkt eines geworfenen Objekts

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Wolfgang Renz
Zweitgutachter: Prof. Dr.-Ing. Hans Peter Kölzer

Abgegeben am 27. August 2009

Iordan Pentchev

Thema der Diplomarbeit

Robotersteuerung in interaktiver Testumgebung: Optimierte Anfahrt an extrapolierten Aufschlagpunkt eines geworfenen Objekts

Stichworte

Robotersteuerung, Tracking, Quadratische Regression

Kurzzusammenfassung

Das im MMLab installierte 3D-Trackingsystem der Firma A.R.T. GmbH zur optischen Bewegungsverfolgung eignet sich sehr gut für die Herstellung einer interaktiven Testumgebung für mechatronische Systeme.
Ziel dieser Arbeit ist die optimierte Anfahrsteuerung eines Roboters an den mittels Bewegungsverfolgung extrapolierten Aufschlagspunkt eines geworfenen Objekts. Der Festo-Robotino folgt einem Fortbewegungsprogramm das unter Echtzeitanforderungen aus Positionsdaten berechnet wird, die das Trackingsystem mit 60 Hz über UDP/IP liefert, wenn ein Ball innerhalb des Trackingvolumens geworfen wird.
Da beabsichtigter Ort und Fahrtrichtung des Roboters von den tatsächlichen Werten wegen des Schlupfs auf der Unterlage abweicht, werden auch die Positions- und Orientierungsdaten des Roboters aufgenommen, um zu einer funktionierenden Bewegungsregelung zu kommen.

Iordan Pentchev

Title of the paper

Robot control in an interactive test environment: Optimal approach to extrapoleted poin of impact of a thrown object.

Keywords

Robot control, tracking, quadratic regression

Abstract

The in the MMLab installed 3D trackingsystem for optical motion-tracking from the company A.R.T GmbH is very suited for creating an interactive test environment for mechatronic systems. The goal of this project is the optimized approach of a robot to the with motion-tracking extrapolated point of impact of a thrown object. When a ball is thrown within the tracking volume, the Festo-Robotino follows a motion program whose real time calculation is based on the robot position data. As the planned location- and direction values of the robot differs from the actual values due to slippage on the basis, the position- and orientation data of the robot are also being recorded in order to achieve an efficient motion control.

Danksagung:

Ich will mich bei meinem Betreuer Professor Wolfgang Renz für die gute Betreuung bedanken.

Außerdem bedanke ich mich bei Professor Hans Peter Kölzer durch dessen Vermittlung dieser Arbeit zustande kam und der meine Arbeit als Zweitkorrektor bewertet hat.

Auch bei allen anderen Leuten die mich während diese Arbeit unterstützt haben.

Andras Ißleib, Steffen Kaufman, Stephan Plaschke, Jan Sudeikat, Thomas Preisler, Jens Zeyn-Kranz, Peter Suchan Assen Popov, Marco Matties, Georg Föhrweißer, Eike Jenning, Torben Albrecht, Frank Siebenmorgen und Janina Wierzoch.

Insbesondere bei Ivan Litchev, der mir den Laptop zur Verfügung gestellt hat auf dem ich diese Arbeit entwickelt und geschrieben habe.

So wie bei meiner Schwester Neli Cadillon, meinem Schwager Jean-Marc Cadillon und auch Rumiana Borisova die mich im letzten Monaten finanziell unterstützt haben.

Ich will mich auch speziell bei meiner Freundin Maria Skov Pedersen, weil sie mich ganze Zeit in vielfältiger Weise unterstützt hat, bedanken.

Alle diese Leute gebührt mein Herzlicher Dank.

Iordan Pentchev Hamburg 25.08.2009

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einführung

Das Wort „Robot“ wurde vom tschechischen Schriftsteller „Karel Cˆapek“ geprägt. In seinem Theaterstück „Rossum’s Universal Robots“ aus dem Jahre 1921 bezeichnete „Robot“ eine Maschine, die dem Menschen das Arbeiten erleichterte. Das Wort „Robot“ ist vom slawischen Wort „Robota“, was zu Deutsch Arbeit bedeutet, abgeleitet.

Es existieren verschiedene Arten von Robotern, die für verschiedene Zwecke entworfen sind. Es gibt Industrie-Roboter, die überwiegend in der Automatisierungstechnik und Produktion eingesetzt werden. Diese Roboter sind dafür konzipiert, präzise und für Menschen gefährliche Arbeit zu bewältigen, auch sind sie oft viel schneller als Menschen.
Desweiteren gibt es Dienstleistungsroboter und Service-Roboter, die in Haushalt, Gesundheit und Pflege eingesetzt werden. Zu dieser Gruppe gehören auch Überwachungs-Roboter und Spielzeug-Roboter. Besonders in Japan sind die Haushalt- und Spielzeug-Roboter z.B. als elektronische Haustiere sehr beliebt.

Schon seit langer Zeit haben Menschen den Traum, menschenähnliche Maschinen zu erschaffen. Humanoide Roboter, die sich menschlich bewegen z. B. auf zwei Beinen gehen können, sind heutzutage ein beliebtes Forschungsthema. Ein Beispiel dafür ist der ASIMO der Firma Honda, welcher sogar in der Lage ist zu joggen. Auch in der Wissenschaft werden Roboter vermehrt eingesetzt. Sie erforschen z. B. das Weltall oder die Tiefen der Ozeane.

Zu dem existieren Roboter, die auch für Lehrzwecke eingesetzt werden können. Ein Beispiel dafür ist „Lego Mindstorm“. Spielwaren Hersteller „LEGO“ hat eine Serie von Baukästen mit programmierbaren Legosteinen RCX(Robotics Command System). Ein anderer Roboter für Ausbildungszwecke ist in der Medizin zu finden. Sein Name ist „SimMan 3G“. An diesen Roboter können die Medizinstudenten(innen)ihr Wissen ohne Risiken praxisnah testen. Ein weiterer Roboter ist von der Firma „Festo“, sein Name ist Robotino. Der Robotino ist ein Roboter, der überwiegend in der Automatisierungstechnik eingesetzt wird. Die Abbildung 1.1 zeigt diesen Robotino-Roboter.

Abbildung 1.1: Robotino

Dieser Roboter wird auch in dieser Diplomarbeit benutzt. Der Robotino ist ein mobiler Roboter, was bedeutet, dass er einen eigenen Antrieb und eine eigene Energieversorgung besitzt. Eine detaillierte Beschreibung dieses Roboters wird im Kapitel „Stand der Technik“ gegeben.

Die vorliegende Arbeit ist eine erste Machbarkeitsstudie mit dem Ziel, unter Nutzung der im MMLab der HAW Hamburg vorhandene interaktiven Testumgebung, einen Roboter einen geworfenen Ball fangen lassen.
Die weiterführende Zielsetzung zukünftiger Arbeiten ist es, Roboter Ball spielen zu lassen. Dabei handelt es sich um die Steuerung von zwei oder mehreren Robotern, die autonom die Position eines fliegenden Balls ermitteln, diesen fangen und wieder per Katapult in eine bestimmte Richtung werfen. Dort wird der Ball von einem anderen Roboter aufgefangen. Danach muss der Roboter selbst entscheiden, in welche Richtung und mit welcher Geschwindigkeit er der Ball wirft. Um voll autonom zu sein, muss der Roboter auch über „Augen“ verfügen. Mit Augen ist gemeint, dass der Roboter mit zwei Kameras ausgestattet ist. Mit diesen Kameras kann er seine Umgebung erkennen und anhand dieser Information selbständig Entscheidungen treffen.

Autonomer Roboter bedeutet, dass der Roboter in der Lage ist die von ihm verlangte Aufgabe ohne fremde Hilfe selbständig durchzuführen. Bezogen auf diese Aufgabe muss der Roboter selbst erkennen wann der Ball geworfen wird, den Ball-Landepunkt berechnen, sich dorthin bewegen und den Ball noch in der Luft fangen.

In dieser Arbeit hat der Roboter keine eigene Kameras, sondern wird in einer Testumgebung mit Daten versorgt.

Das Themengebiet ist komplex und beinhaltet nicht nur Teile der Elektrotechnik oder Informatik. Zusammen mit der Informationstechnik und Elektrotechnik ist auch die Mechanik eingebunden. So ein komplexes System das die oben genannten Gebiete beinhaltet kann, wird als Mechatronisches System bezeichnen.
Das Wort Mechatronik ist ein Kunstwort, das im Jahr 1969 von der Japanische Firma Yaskawa-Electric Corporation benutzt wurde. Mechatronik ist ein interdisziplinäres Gebiet bei dem folgende Disziplinen zusammenwirken:

- Mechanik (Maschinenelemente, Maschinen, Feingerätetechnik)
- Elektrische Systeme (Mikroelektronik, Leistungselektronik, Messtechnik, Aktorik)
- Informationstechnik (Systemtheorie, Regelung- und Automatisierungstechnik, Software-Gestaltung)

Heute verschmelzt das Gebiet Mechatronik die Disziplinen Informatik (Informationstechnik), Elektrotechnik und Mechanik mit einander. Meist werden die Mechatronischen Systeme mittels Regelkreisen beschrieben um sie leichter regeln zu können.

Das System als Ganzes betrachtet ist mechatronisch (Isermann, 2008).

> Integrierte mechanische-elektronische Systeme entstehen durch eine geeignete Kombination von Mechanik, Elektronik und Informationsverarbeitung. Dabei beeinflussen sich diese Bereiche wechselseitig. Man beobachtet zunächst eine Verlagerung von Funktionen der Mechanik zur Elektronik, dann die Hinzunahme von erweiterten und neuen Funktionen und schließlich entwickeln sich Systeme mit gewissen intelligenten bzw. autonomen Eigenschaften. Für dieses Gebiet der integrierten mechanisch-elektrotechnischen Systeme wird seit einigen Jahren der Begriff „Mechatronik“ verwendet.

Das folgende Abbildung zeigt der Zusammenhang eines mechatronischen Systems.

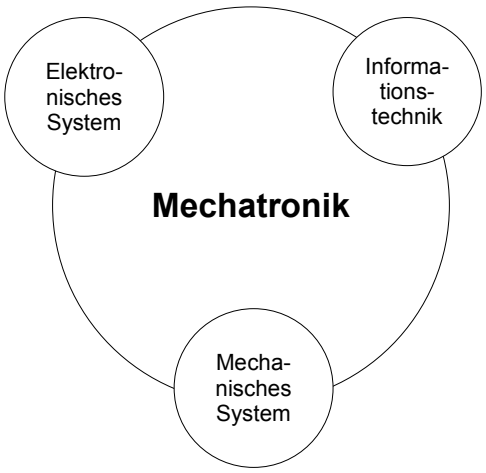


Abbildung 1.2: Zusammenhang ein mechatronisches Systems (Isermann, 2008)

Die Bewegungen dieses Roboters müssen erfasst werden. Das geschient mittels optisches Tracking[1]. Ein solches tracking System von der Firma ART-GmbH[2] wird auch in dieser Diplomarbeit benutzt. Das ART-System wird detailliert in dem Kapitel-2 „Stand der Technik“ beschrieben. ART ist die Abkürzung von (Advanced Realtime Tracking).

Die Auswertung und die Simulation wird mittels Matlab[2] durchgeführt. Matlab ist ein kommerzielles Softwareprodukt, das überwiegend zu numerischen Berechnungen mittels Matrizen benutzt wird. Der Einsatz dieser Software ist sehr weit verbreitet und kann auch zu Analysieren von Testdaten benutzt werden. Das ART-System und Matlab bilden eine „interaktive Testumgebung“. Mittels dieser können die Bewegungen des Roboters von ART-System erfasst und dann mittels Matlab analysiert werden.

Das ART-System besteht aus vier Kameras und einem Tracking-Rechner mit einem Timedelay von $20\ ms$. Zu dem ART-System gehören auch Bodies.[3] Die Bodies werden von Kugeln, die mit einer reflektierenden Folie bedeckt sind, konfiguriert. Diese Kügelchen werden starr mit einander verbunden, so bilden sie einen Solid-Body[4]. Diese sind in verschiedenen 3D-Konfigurationen aufgestellt, so dass jeder einzelne Body eine einzigartige Kugel-Konfiguration aufweist. Auf diese Weise kann nach Kalibrierung jeder einzelne Body zu jedem Zeitpunkt vom ART System identifiziert und verfolgt werden.

[1] Ist Bestimmung der Position und die Orientierung ein bewegtes Objekt in bestimmte Zeitintervalle. Meistens bestehen diese Systemen aus zwei oder mehrere Kameras um eine 3d Erfassung des bewegten Objektes zu ermöglichen

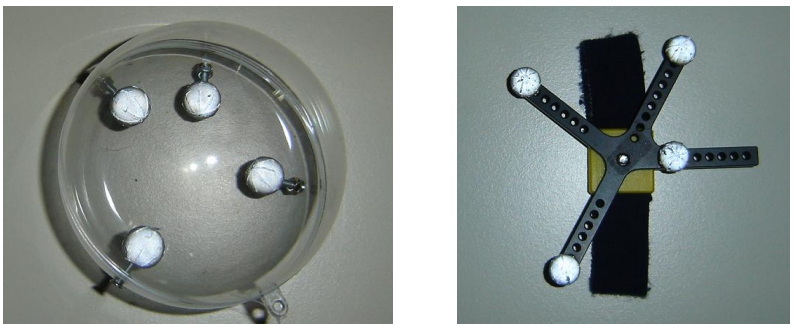
[2] http://www.ar-tracking.de/

[2] http://www.mathworks.de/

[3] Kugelförmige, reflektierende Marker zur Erkennung des vom ART-System zu verfolgenden Objektes

[4] Feste Körper

In folgender Abbildung sind beide Erkennungs-Bodies dargestellt, die für den Roboter-Robotino und den Ball benutzt werden.



(a) Body für den Ball (b) Body für den Roboter

Abbildung 1.3: ART Bodies

Das Gerüst des ART-Systems in MMLab des HAW-Hamburg ist in folgender Abbildung zu sehen. Die vier Kameras sind so ausgerichtet, dass immer zwei ein Objekt innerhalb des Tracking-Volumens erfassen können. Der Tracking-Rechner ist auf der Abbildung nicht zu sehen. Eine Abbildung dieses Rechners und eine detaillierte Funktionsweise wird in der Kapitel „Stand der Technik“ geben.

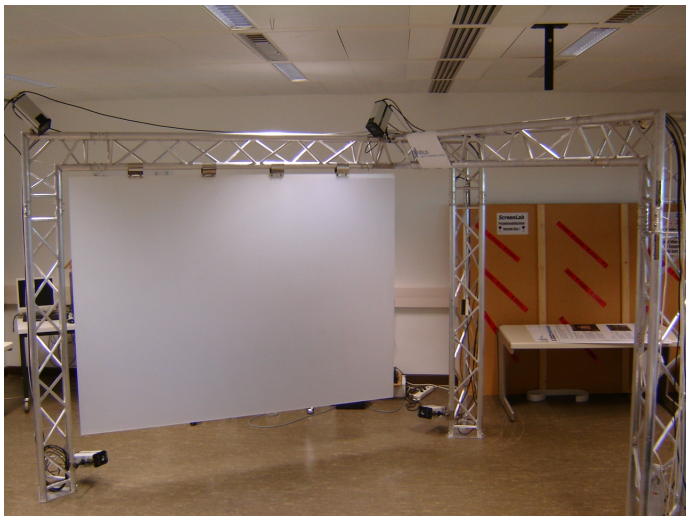


Abbildung 1.4: Das Gerüst das ART-System im MMLab der HAW-Hamburg

In der Themenstellung wurde die Systemarchitektur sowie der Roboter und das ART-System, das zusammen mit der Auswertung in Software Matlab zum Aufbau einer interaktiven Testumgebung für den mobile Roboter dienen soll, vom Betreuer, dem Leiter des MMLab

vorgegeben. Zu der Testumgebung gehört auch der Ball (siehe Abbildung 1). Er ist aus zerbrechlichen Kunststoff gebaut, so dass wenn er auf dem Boden fällt zerbricht. Weiter Beschreibungen sind in dem Kapitel „Technische Analyse“ zu finden. Das ART-System muss für die Anwendung konfiguriert und kalibriert werden. Zu dem ganzen System muss auch der Steuer-Rechner noch eingefügt werden. Der Steuer-Rechner ist ein ThinkPad-Lenovo Notebook, mit Dual Core Prozessor-2.4 GHz. Mit diesem Rechner werden alle Berechnung- und Steuer-Programme entworfen und ausgeführt.

1.1 Motivation

In dieser Diplomarbeit muss das Zusammenspiel der Komponenten eines mechatronisches System in realen Einsatz untersucht werden.

Um die Diplomarbeit durchzuführen und mit Erfolg abzuschließen, müssen folgende Untersuchungen gemacht werden:

- Ist der Roboter schnell genug um den Ball noch in der Luft zu fangen.
- Ist es möglich auf dem Bodenbelag(Fußboden in MMLab) den Roboter mit höher Geschwindigkeit stabil zu fahren.
- Die Reaktion des Roboters auf neue Instruktionen in Echtzeit (wie schnell kann der Roboter seine Geschwindigkeit und Richtung verändern).
- Der Beschleunigungs- und Bremsvorgang des Roboters bei verschiedenen Geschwindigkeiten anhand Geschwindigkeitsänderung zu untersuchen. Ist die Reibungskraft, die auf die Roboter-Rädern wirkt, groß genug, um die gewünschte Beschleunigung und Bremmsverhalten zu erzielen.
- Ist Windows-XP als Betriebssystem schnell genug für Empfang von UDP/IP Pakete von dem Tracking Rechner, die Berechnung durchzuführen und als Steuerrechner zu fungieren.
- Ein Ball mit einem Identifikation-Body als zu werfendes Objekt zu bauen.
- Für den Roboter eine Fangeinrichtung(um der Ball zu fangen) zu konstruieren und auf den Roboter zu befestigen.
- Wie gut wird ein fliegender und rotierender Ball (siehe Abbildung1) vom ART-System getrackt.
- Eine Testumgebung für Wurfversuche aufbauen damit der Ball nicht zerbricht und der Roboter nicht beschädigt wird.

- In diesem System ist es nicht möglich alles analytisch zu berechnen, deshalb muss ein Regelkreis für das ganze System entworfen werden.

1.2 Aufgabenstellung

In dieser Diplomarbeit muss untersucht werden, ob es möglich ist, mit dem Robotino unter Verwendung des ART-Systems als interaktive Testumgebung den geworfenen Ball (siehe Abbildung 1) noch in der Luft zu fangen, wenn der innerhalb des vom ART-Systems gedeckte Trackingvolumens parabelförmig geworfen wird. Das ART-System erfasst die Position des Balles und die des Robotinos. Anhand der vom ART-System gelieferten Ballkoordinaten, wird die Ball-Landeposition berechnet. Durch die aktuelle Robotino Position, die auch vom ART-System getrackt wird, und die schon berechnete Ball-Landeposition wird die Robotino Geschwindigkeit auf X- und Z-Achse sowie die Fahrzeit des Roboters berechnet. Der berechnete Ball-Landepunkt ist die Soll-Position, zu dem der Roboter fahren muss, die aktuelle Position des Roboters ist die Ist-Position. Weicht der Roboter von der vorgegebenen Richtung ab (z.B. durch Schlupf der Rädern), wird er durch einen Regelalgorithmus wieder in die richtige Richtung gebracht. Diese so beschriebene Aufgabenstellung wird modular abgearbeitet.

Hier muss die Echtzeit für diese Aufgabe definiert werden. Unter Windows-XP ist es nicht möglich echtzeitfähige Programme laufen zu lassen, denn es müssen immer die selben Zeiten eingehalten werden. Hier ist zu erwähnen, dass wenn in dieser Diplomarbeit über Echtzeit geschrieben wird, ist quasi Echtzeit gemeint, die mit gewissen Zeit-Toleranzen(Jitter-Verhalten) rechnen muss.

- Die Koordinaten des Balles aus dem ART-System einlesen.
- Daraus den voraussichtlichen Landepunkt des Balles ermitteln.
- Daraus in Echtzeit neue Zieldaten(Geschwindigkeit auf X- und Z-Richtung sowie die Fahrzeit) für den Roboter “Robotino“ errechnen.
- Der voraussichtliche Ball-Landepunkt sowie die Roboter Geschwindigkeit und Roboter Fahrzeit werden ständig und permanent in Zeitintervallen von ca. $20\ ms$ berechnet und bis zum Zeitpunkt des Aufschlags zum Roboter via WLAN übertragen.
- Ziel dieser Diplomarbeit ist es, dass der Roboter den Ball noch in der Luft fängt.

Der Roboter wird mit eigenem Fangkorb ausgestattet, in den der geworfene Ball fallen soll.

Die Software soll in der Programmiersprache C++ erstellt werden. Die Auswertung der Versuche werden innerhalb der interaktiven Testumgebung durchgeführt.

An der Standard-Softwareausstattung des Systems: ART und Robotino werden keine Änderungen vorgenommen. Die Datenkommunikation mit dem Roboter erfolgt über die Funktionen von Robotino-API[5]. Die Kommunikation mit dem ART System erfolgt über die UDP/IP Protokoll-Familie. Das ART-System muss für diese Aufgabe kalibriert werden und die Kameras des ART-Systems so ausrichten werden, dass ein größtmögliches tracking Volumen entsteht.

1.3 Vorgehensweise

Diese Arbeit kann in folgende Schritte eingegliedert werden:

- Konstruktion eines durchsichtigen Balles, der auch mit Id(Erkennung) ART-Body im inneren des Balles ausgestattet (Konfiguriert) wird.
- Mathematisch-Physikalische Analyse des fliegenden Balles. Ziel ist es, ein Mathematisches Modell zu entwerfen mit dem eine gute Vorhersage des Ball-Landepunktes berechnet werden kann.
- Steueralgorithmus für den Roboter zu entwerfen. Ziel ist es, seine Stabilität zu erhöhen bezüglich der größtmöglichen Geschwindigkeit. Der Schlupf der Roboter-Räder auf dem Fußboden experimentell zu minimieren.
- Das System als ein mechatronisches System betrachten und einen Regelungstechnischen Plan erstellen.
- Design der Software auf Grund des schon erstellten Regelungstechnischen Plans.
- Implementation der Software.
- Testen und Analysieren des Verhalten des Systems
- Reverse Engineering. Aus den Analysen der Tests, das System verbessern.
- Verbesserungen vorschlagen, wenn es welche gibt.

Diese Diplomarbeit wurde Stufenweise Aufgebaut, da es sich um eine erste Machbarkeitsstudie handelt. In gewissen Zeitphasen, während der Entwicklung, mussten nach Wegen gesucht werden, wie weiter vorzugehen ist. Der theoretische Entwurf könnte nicht direkt und unproblematisch umgesetzt werden. In den nächsten Kapiteln wird detailiert auf die Schwierigkeiten eingegangen.

[5]ist eine von Festo-Didaktik zur Verfügung gestellte kostenlose Bibliothek um der Programmierung des Robotinos zu ermöglichen

2 Stand der Technik

In diesem Kapitel wird das Gesamtsystem mit allen seinen Komponenten vorgestellt, so wie deren technischen Daten erläutert. Zusätzlich wird es eine kurze Übersicht von vergleichbaren Systemen geben.

2.1 Mechatronische Systeme

Wie in dem Kapitel Einführung schon erwähnt wurde, ist das Wort Mechatronik ein künstliches Wort. In den letzten Jahren entwickelt es sich als eigenständige Ingenieurwissenschaft. Es ist ein interdisziplinäres Gebiet, das sich auf die drei Disziplinen Maschinenbau, Elektrotechnik und Informatik stützt. Ein typisches mechatronisches System bekommt als Eingänge Signale. Diese werden verarbeitet und steuern die Energieumwandlung und/oder Versorgung des Systems mit Energie. (Roddeck, 2002)

> Bei mechatronischen Systemen erfolgt die Lösung einer Aufgabe sowohl auf mechanische als auch digital-elektronischem Wege. Hierbei spielt die Wechselbeziehungen bei der Konstruktion eine Rolle. Während bei einem konventionellen System sowohl der Entwurf als auch die räumliche Unterbringung der mechanischen und elektronischen Komponenten getrennt sind, zeichnet sich ein mechatronisches System dadurch aus, dass das mechanische und elektronische System von Anfang an als räumlich und funktionell integriertes Gesamtsystem zu betrachten ist. Dann wird die Gestaltung des mechatronischen System schon beim Entwurf von auch vom elektronischen System her beeinflusst. Ein weiteres Merkmal mechatronischer Systeme ist die integrierte digital Informationsverarbeitung. (Isermann, 2008)

Ein mechatronisches System verfügt über eine eigene Energieversorgung, es kann auch zusätzliche Energie von außen in das System eingeführt werden. In der Abbildung 2.1 ist die Funktionsweise eine mechatronisches System bezeichnet:

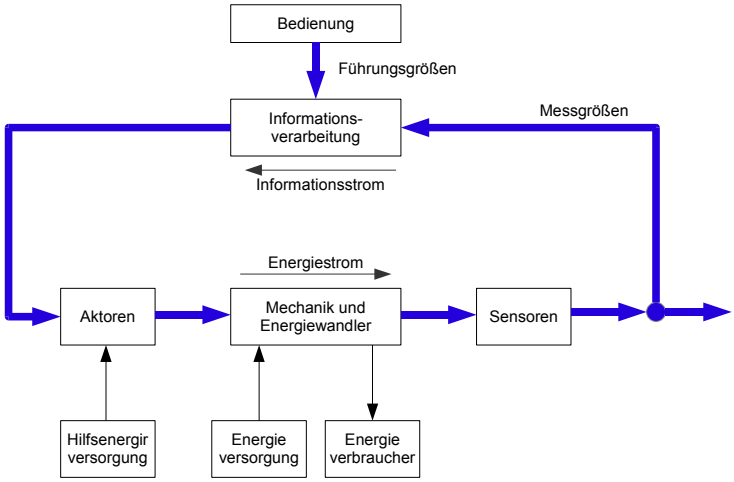


Abbildung 2.1: mechatronisches System
(Isermann, 2008)

Durch die Schnittstelle „Bedienung“ wird das System initialisiert. Die Steuerung oder Regelung Größen werden in dem System bekannt gemacht. In der „Informationsverarbeitung“ werden die Signale, die von den Sensoren gemessen werden, zusammen mit den Steuerungs und Regelungsgrößen der gemessenen Signale verarbeitet. Auf diese Weise wird das System gesteuert und geregelt, damit es immer in einem stabilen Zustand bleibt. Die Aktoren sind das Gegenteil der Sensoren. Diese sind aktive Elemente wie elektrische oder pneumatische Ventile oder Relais so wie mikroelektronische integrierte Schaltkreise. Die Aktoren sind der aktive Teil in dem System. Sie regeln/steuern den elektrischen oder pneumatischen Fluss. Sie schließen, öffnen elektrische oder pneumatische oder mechanische Kreise. Mit deren Hilfe wird auch z.B. der Regelung von einer Elektromotor, der eine bestimmte Geschwindigkeit unabhängig von die Belastung(in vorher bestimmter Bereich) halten muss.

> Die Aufgabe von Aktoren ist die Umsetzung von leistungsarmen Stellgrößen (0-10 V oder eingeprägte Ströme 0-20 mA) in Prozesseingangsgrößen mit einem in der Regel wesentlich höheren Leistungsniveau. (Isermann, 2008)

Mit deren Hilfe wird auf Grund der berechnete Größen in der System-Komponente „Informationsverarbeitung“ das ganze System geregelt damit es in einem stabilen Zustand gehalten wird. In der System-Komponente „Mechanische Energiewandler“ wird Energie umgewandelt. Es handelt sich in den meisten Fällen um Elektromotoren, die elektrische in mechanische Energie unter Überwachung durch die Steuerelektronik umwandeln. Die Sensoren beobachten und messen die Änderungen in dem System und geben diese weiter zu Verarbeitung (A/D Wandler). Durch die Digitalisierung der gemessenen Signale ist es möglich mit dem Computer diese mit sehr große Genauigkeit zu verarbeiten (z.B. Rauschen unterdrücken, addieren, etc.).

Durch die Schnittstelle „Bedienung“ werden die erforderlichen Führungsgrößen in das System eingeführt, die bei der Initialisierung, Konfiguration etc. benötigt werden. In Aktoren in dem System kann Energie von außen zugefügt werden. In so einem System wird eine Energieumwandlung stattfinden. Wo und in welche Richtung hängt von dem System ab, z.B. elektrische in mechanische Energie bei Elektromotoren. Sensoren erfassen die Daten an bestimmten Stellen, die Aktoren leiten das System damit dieses geregelt werden kann.

2.2 Mobile Serviceroboter

Ein Serviceroboter ist eine freiprogrammierbare autonome Bewegungseinrichtung, die vollautomatische Dienstleistungen in einer vorher bekannten Umgebung an Menschen oder Einrichtungen verrichtet. (siehe (Rolf Dieter Schraft, 1996))

Ein Mobiler Serviceroboter ist ein Serviceroboter, der seinen Standort durch Lokomotion verändern kann. Der mobile Roboter bewegt sich in der bekannten Umgebung in der er sein Servicetätigkeiten verrichten kann. (siehe (Nehmzow, 2002))

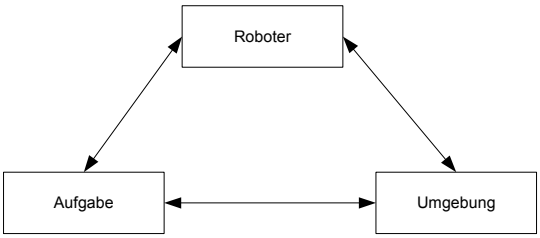


Abbildung 2.2: mechatronisches System
(Nehmzow, 2002)

Ein mobiler Serviceroboter besteht allgemein aus einem mobilen und ein oder mehreren Anwendungsmodulen. Die Art der Fortbewegung und die Anwendungsmodule sind für eine bestimmte Umgebung konstruiert. Die Anwendungsmodule können ein oder mehrere Handhabungsarme sein, Messeinrichtungen, Fangeinrichtungen, Schweißeinrichtung etc. Damit ergibt sich die Kinematik des Roboters. Durch die Endeffektoren[1] kann der Roboter die Objekte in seiner Umgebung manipulieren und bearbeiten z.B. mit Werkzeugen. Die mobile Plattform eines Serviceroboters enthält entsprechend seiner Bauweise und Konstruktion verschiedene Eigenschaften der Mobilität[2], er kann schweben, fliegen, schwimmen. Die Art der Fortbewegung oder Fortbewegungen wird von der Umgebung wo der Roboter seinen Service verrichtet bestimmt. (siehe (Rolf Dieter Schraft, 1996))

[1] Greifsysteme
[2] z.B. Räder, Ketten

Vor 45 Jahren wurde in der Industrie weltweit den ersten Roboter bei „General Motors“ installiert. Serviceroboter sind heute im Einsatz in Produktion, Haushalt, Forschung etc..

Serviceroboter im Einsatz[3]:

- Haushalt: Staubsaugerroboter „Trilobite“ von Elektrolux. Dieser Roboter ist in der Lage zu reinigen in einer chaotischen Umgebung per Zufallbahnen oder durch Wandverfolgung. Bevor der Akku ganz leer ist, schaltet der Roboter den Sauger aus, sucht eine Wand und fährt bis zur Dockstation zurück. Das System lädt sich auf. Dann wird die Reinigung fortgesetzt (siehe (Rolf Dieter Schraft, 2004)). RoboCleaner von Kärcher ist auch eine Staubsaugerroboter, der eine besondere Dockstation besitzt. Diese Dockstation ist mit einem Infrarot- Leitstrahl ausgestattet, der den Roboterstaubsauger navigieren kann. Der RoboCleaner kann seine Geschwindigkeit anpassen abhängig von dem Schmutz auf dem Boden (siehe (Rolf Dieter Schraft, 2004)). Der Nass-Reiniger Auto-Cleaner von „PS Automation“, ist auch ein Reinigungsroboter, der eine besondere Strategie zur Reinigung hat. Der Roboter fährt immer spiralförmig um einem vorher berechneten Punkt im Raum. Stößt der Roboter auf ein Hindernis, berechnet er einen neuen Punkt und beginnt um diesen Punkt seine Reinigungsprozedur von Neuem. (siehe (Rolf Dieter Schraft, 2004)) Die Firma „Husqvarna“ hat einen autonomen Rasenmähnroboter entwickelt, der ähnlich wie der Staubsaugerroboter funktioniert. Eine manuell verlegte Begrenzungsschleife markiert den zu bearbeitenden Rasenbereich. (siehe (Rolf Dieter Schraft, 2004)) Eine enorme Leistung mit kleinen Maschinen, die dank der schnellen Entwicklung von Mechatronik zustandekommt, sind die Fensterputzroboter. Der Roboter Raccon ist auch ein Fensterputzroboter, der durch das Fraunhofer Institut entwickelt wurde und für den Haushalt besonders geeignet ist. Dieser ist ein multifunktionaler Reinigungsroboter für senkrechte und geneigte Flächen: Er funktioniert nach dem Prinzip Halten, Bewegen und Reinigen. Im Fraunhofer Institut und auch weltweit ist das Potenzial der Reinigugngsroboter schon längst erkannt worden. (siehe (Rolf Dieter Schraft, 2004)) „Mosro-1“ von der Firma Robowatch ist ein Überwachungsroboter. Dieser Roboter ist mobil und besitzt 240 verschiedene Sensoren von Bewegungs- bis Geruchssensoren. Die Betriebsdauer mit einer Ladung der Batterie ist auf 18 Stunden begrenzt. Dieser Roboter kann selbständig auf sein bewachtes Gebiet mit einer Geschwindigkeit von $4\ \frac{km}{h}$ navigieren. Mit seiner Kamera kann er Personen entdecken und diese auf 24 Sprachen auffordern, sich durch Fingerabdruck zu identifizieren (siehe (Rolf Dieter Schraft, 2004)). „Care-O-bot-2“von dem Fraunhofer Institut hilft mobilitätseingeschränkte Personen in Alltag. Durch Spracherkennung kann er Aufgaben erledigen. Er besitzt einen Handhabungsarm mit sechs Freiheitsgraden und eine Reichweite von einem Meter. Seine Umgebung wird vorher in einer Datenbank gespeichert. Durch Kameras auf dem Kopf und einen Laserscanner kann er die Objekte in der Umgebung mit diesen in der Datenbank vergleichen und so selbständig die Aufgaben im Haushalt erledigen. (siehe

[3]Die folgenden Beschreibungen sind teilweise aus dem Buch von (Rolf Dieter Schraft, 2004) entnommen

(Rolf Dieter Schraft, 2004))

- Inspektion: Inspektionsroboter vermessen Systeme auf verschiedene Arten, um deren Istzustand richtig zu beurteilen. „Makro“ ist ein Sechs-Segment-Abwasserroboter, der mittels Sensoren die Schäden in Abwasserkanälen erkennen kann. Dieser kann die Kanäle auch automatisch vermessen. (siehe (Rolf Dieter Schraft, 2004)) „Theseus-1“ ist ein Miniroboter, der Gasrohre von 25- bis 200 mm Durchmesser überprüfen kann. Dieser Roboter überprüft die Gasrohre durch das Aufbauen eines vorgeschriebenen Drucks. (siehe (Rolf Dieter Schraft, 2004)) „Marvin“ ist eine Minikopter-Roboter, der an der TU-Berlin entwickelt wurde. Dieser hilft bei Waldbränden, Chemieunfällen und Kernkraftwerk-Katastrophen, damit die Menschen nicht unnötigen Risiken ausgesetzt sind.
- Landwirtschaft: Durchschnittlich versorgt ein moderner Landwirt heute etwa 120 Menschen mit Nahrungsmitteln. In den Fünfziger Jahren des letzten Jahrhunderts waren es nur zehn Personen, die ein Landwirt mit Nahrungsmitteln versorgt hat. „Mechanical Weed Control“, ist ein mobiles Robotersystem zur automatischen Bekämpfung von Unkraut. Dieses System ist noch im Testzustand in Schweden. „Roboter-Rover“ ist ein Roboter, der Gänse hütet. Das System besteht aus externen Kameras, die die Bewegung des Roboters und der Gänse verfolgt. Die Kameras senden die Information an eine Rechnereinheit, die die berechneten Daten an den Rover sendet. Das Ziel ist, den Gänseschwarm an ein bestimmten Ort zu manövrieren. (siehe (Rolf Dieter Schraft, 2004))
- Unterwasser: „Spider“ von Cybernetix ist eine Unterwasserraupe, die zur Inspektion von Pipelines und Unterwasserkabeln eingesetzt wird. Es gibt zwei Varianten: ferngesteuert oder voll selbständig, die mittels Bildverarbeitung funktioniert. (siehe (Rolf Dieter Schraft, 2004))
- Weltraum: Neben den traditionellem Robotern, die der Form des Mars-Rover ähneln, entwickelt Nasa derzeit „Snakebot“ aus vielen identischen und mit Gelenken verbundenen Segmenten. Diese Art von Roboter hat enorme Vorteile im Vergleich mit herkömmlichen Robotern. Die „Snakebots“ können Hindernisse leicht überwinden. So können sie fremde Planeten erkunden (siehe (Rolf Dieter Schraft, 2004)).
- Assistenzroboter: Der Assistenzroboter „rob@work“ ist ein in der Produktion einsetzbarer Helfer am Arbeitsplatz. Dieser Roboter ist eine selbständig navigierende Plattform mit einem Roboterarm. Dieser Roboter soll den Menschen bei schwierigen Aufgaben wie Kleben oder Schweißen als dritter Arm helfen. (siehe (Rolf Dieter Schraft, 2004))
- Medizin: „Evolution-1“ der Firma Universal Robot Systems(URS) ist ein System mit Vorpositionierungskinematik. Das System nutzt einen Hexapod zur Positionierung von Instrumenten. Er wird in der Endoskopie benutzt. (siehe (Rolf Dieter Schraft, 2004))

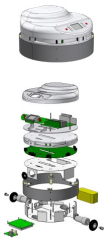
- Edutainment[4]: Besonders in Japan sind diese Art von Mobilen Robotern sehr verbreitet. „Aibo“ von Sony ist ein elektronischer Spielzeughund, der eine sehr große Popularität genießt und schon in der dritten Generation auf dem Markt ist. Auf speziellen Aibo-Veranstaltungen treten internationale Fußballmannschaften gegeneinander an. Papero von NEC ist ein Unterhaltungroboter, der auch durch Streichelsensoren Gefühle wie Lob und Zuneigung empfinden kann. (siehe (Rolf Dieter Schraft, 2004))
- Humanoiden: Die mobilen Service-Roboter werden nach menschlichem Vorbild gebaut. Dank der Mechatronik entwickelt sich dieser Bereich sehr schnell. Honda begann 1986 mit dem Bau der ersten Laufmaschinen. Die erste Laufmaschine namens „E0“ war sehr langsam. Es folgten die „E1“ bis „E6“. Die Laufmaschine „E6“ konnte schon stabil auf Stufen einer schiefen Ebene laufen. Mit „Asimo“ hat die Firma „Honda“ auf dem Gebiet der Mobilität auf zwei Beinen große Fortschritte gemacht. Der „Asimo“ kann schon joggen und nicht nur auf gerader Ebene. Mit eingebauter Sprach- und Gestik-Command-Steuerung ist „Asimo“ sehr geeignet als Service-Roboter, der autonom bestimmte Handlungen ausführen kann. Sony hat einen kleinen Humanoid-Roboter namens „QRIO“ entwickelt. Der 0.6 m kleine Zwerg soll als Unterhaltung bei Präsentationen von Sony-Produkten eingesetzt werden. Er wird nicht im Handel erhältlich sein. Dieser kleine Entertainer kann auch tanzen, singen, von außen Kräfte abfangen und wenn er auf den Boden gefallen ist, kann er von selbst wieder aufstehen. Er ist in der Lage, dank seines enormen Wortschatzes von ca. 60000 Wörter ein Smalltalk mit Menschen zu führen. (siehe (Rolf Dieter Schraft, 2004))

Robocup ist eine internationale und weltweit aktive Bildungs- und Forschungsinitiative, die es sich zum Ziel gesetzt hat künstliche Inteligenz und Roboterforschung zu fordern.

2.2.1 Roboterplattformen für die Ausbildung

Simulation spielt eine zentrale Rolle bei der Entwicklung von Ausbildungssysteme. Aus dieser Sicht haben mobile Roboterplattformen für Ausbildung und Forschung eine enorme Bedeutung. Um Algorithmen aus unseren realen Welt zu testen und zu optimieren bitten einige Firmen den Wissenschaftlern Versuchs-Plattformen. Hier werden einige Roboter für für die Ausbildung vorgestellt:

[4] unterhaltsames Lernen

Name	Hersteller	Bild	Beschreibung
Pekee-2	Wany Robotics	 (Wany)	Die Firma bietet der Roboter in drei Variante an: Essential,Advanced und Ultimate
Khepera-2	K-team	 (K-Team)	• Breite: 0.07 m • Höhe: 0.03 m • Gewicht: 0.08 kg • Nutzlast: 0.25 kg
AmigoBot	Activrobots	 (activrobots)	• Länge: 0.33 m • Breite: 0.28 m • Höhe: 0.15 m • Gewicht: 3.6 kg • Nutzlast: 2 kg
PowerBot	Activrobots	 (activrobots)	• Länge: 0.85 m • Breite: 0.63 m • Höhe: 0.47 m • Gewicht 120 kg • Nutzlast 60 kg
Robotino	Festo		• Gewicht: 11 kg • Radius: 0.36 m • max.Geschwindigkeit: 10 $\frac{km}{h}$ • Nutzlast: unbekannt

Keiner dieser oben genannten Roboter außer der Robotino verfügt über ein Omnidrive-Antriebs-System. Dadurch werden dessen Fähigkeiten zu manövrieren eingeschränkt.

Ein weiterer Roboter, der auch zu Ausbildungs- und Forschungszwecken verwendet wird, ist der „Hoap“ von der Firma „Fujitsu“. Dieser Roboter kann zum Lernen humanoider Bewegungen gebracht werden. Diese Art von Roboter sind in letzter Zeit sehr populär geworden z.B. in „RoboCup“, wo diese in Mannschaften Fußball spielen.

Der Roboter „Hoap“ ist etwa 0.5 m hoch und wiegt etwa 6 kg. Er wird zusammen mit einer Simulationssoftware ausgeliefert. Mit dieser mobilen-Plattform können Algorithmen für die Bewegung zweibeiniger Roboter entwickelt werden. „Fujitsu“ hat eine Schnittstelle für die HOAP-1 Architektur zur Verfügung gestellt. Damit können eigene Programme entwickelt und getestet werden.

Abbildung 2.3: HOAP-1 von Fujitsu
(Fujitsu)

Der Roboter, der aus dem Hause „Festo“ stammt, hat den Namen „Robotino“ (siehe Abbildung-1.1). Dieser Roboter ist auch auf einer mobilen Plattform aufgebaut. Der Unterschied zwischen dem Robotino und allen bis jetzt erwähnten mobilen Robotern ist, dass der Robotino einen Omnidrive-Antrieb besitzt. Abbildung 2.4 zeigt die Anordnung der Räder bei einer Omnidrive Antriebssystem.

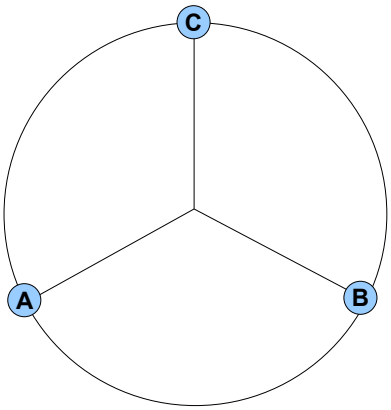


Abbildung 2.4: Omnidrive Antrieb Rädern Anordnung

Die Abbildung 2.5 zeigt das Rad mit dem Elektromotor. Die Transmission des Robotinos beträgt eine Übersetzung von 19:1.

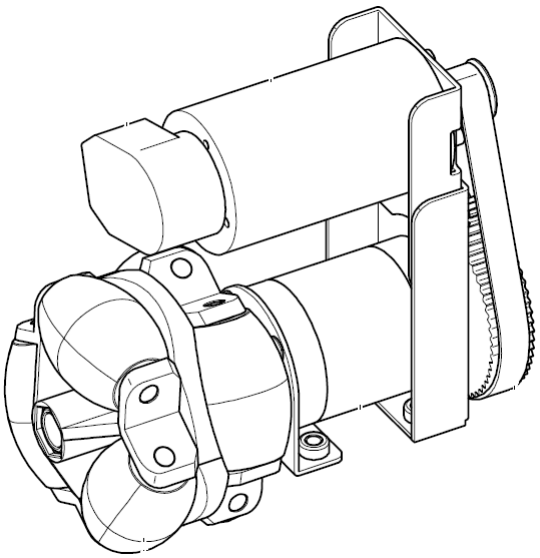


Abbildung 2.5: Robotino Omnidrive Antrieb für das Rad
(Festo)

Auf der Abbildung 2.6 ist der Robotino ohne Plastik-Deckel zu sehen. Zu sehen ist die sternförmige Anordnung der Räder und die Elektromotoren (Abbildung 2.4). Mit den Buchstaben A,B,C sind die Räder und die dazugehörigen Elektromotoren markiert.

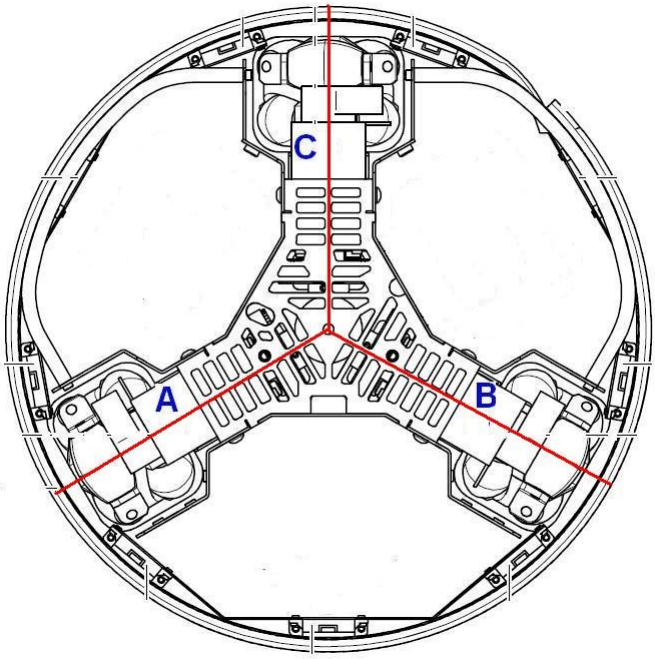


Abbildung 2.6: Omnidrive Antrieb Robotino ohne den Deckel
(Festo)

Der Omnidrive Antrieb des Robotinos ermöglicht es Kurven zu fahren ohne vorher eine Drehbewegung durchzuführen. Diese Eigenschaft des Omnidrive-Antriebssystems ist sehr wichtig bei Echtzeitanwendungen. Die Zeit für die Drehbewegung wird gespart und so ist der Roboter schneller als mit einem herkömmlichen Antriebssystem von einer Stelle zu einer anderen fahren zu lassen.

Der Roboter Robotino wurde für diese Aufgabe mit einem ART-Body und einem Fangkorb ausgestattet. In Abbildung 2.7 ist der Robotino mit dem Fangkorb und dem ART-Body dargestellt. Der Fangkorb ist rund und hat einen Durchmesser von 0.32 m. Der ART-Body befindet sich mittig 0.12 m über dem Fangkorb. Auf diese Weise ist der ART-Body in der Mitte des Roboters plaziert, damit dieser von allen vier Kameras detektiert werden kann. Wenn der Body auf der Höhe des Fangkorbrandes befestigt würde, würde der ART-Body nicht von den unteren zwei Kameras detektiert werden.

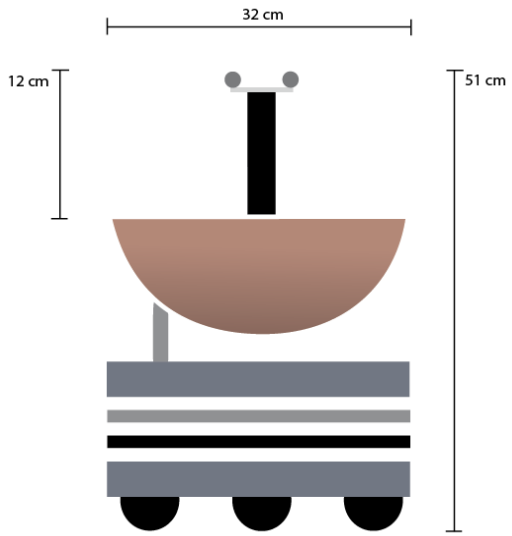


Abbildung 2.7: Robotino mit dem Fangkorb und Identifikationsbody

Weitere technische Daten der Robotino befinden sind im Appendix.

2.3 Interaktive Testumgebungen

Die Testumgebung ist eine Einrichtung um die Zuverlässigkeit von Produkten, Anlagen, Soft-/Hardware etc. zu testen. Die Abbildung 2.8 zeigt den prinzipiellen Aufbau einer Testumgebung. In vielen Fällen wird keine Interaktion in die Testumgebung eingebaut, durch die die zu testenden Anlagen mit Daten versorgt werden. Die Ergebnisse werden nach Ablauf des Testes automatisch oder manuell abgeholt und anschiessend ausgewertet.

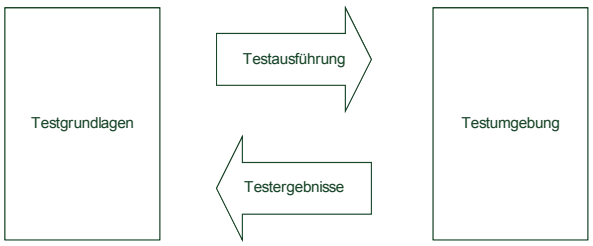


Abbildung 2.8: Testumgebung

In einigen Fällen wird das Testobjekt während der Testphase in Bewegung gesetzt. Es wird dann eine Interaktion gebraucht, um die neue Position und den Zustand des zu testenden

Objektes zu prüfen. Die Testergebnisse müssen automatisch in Echtzeit verarbeitet werden. Die Analyse des Testes kann auch anschliessend erfolgen und muss nicht mehr in Echtzeit erfolgen. Die älteste interaktive Testumgebung ist der Kompass. Der Kompass liefert die Richtung in Echtzeit zurück. In der Interaktiven Testumgebungen handelt es sich meistens um Bewegungsverfolgung (Motion Tracking). Eine interaktive Testumgebung wird auch in dieser Diplomarbeit benutzt. Diese besteht aus dem kommerziellen Advanced Real Tracking System (ART) und einem speziell präparierten Ball sowie einem Positionsmarker für den Roboter. Matlab dient als Testsoftware, um den Test zu analysieren.

2.3.1 Motion Tracking, optische Bewegungsverfolgung

Unter Motion Tracking versteht man Systeme mit deren Hilfe die Position und Ausrichtung eines Objekt (Person, Tiere, Gegenstand, Roboter) in einem fest definierten 3D Raum ermitteln. Diese Technik wird eingesetzt um Bewegungen von Menschen oder Tiere im virtuellen Raum darzustellen. Desweiteren können Bewegungen von Objekten um die genaue Position dieser Objekte in Echtzeit zu berechnen benutzt werden.

Es existieren verschiedenen Techniken um Motion Tracking zu realisieren.

Motion Tracking Systeme können in viele Gruppen unterteilt werden siehe (Renz, a):

- Elektromechanische Systeme
- Akustische Systeme
- Optische Systeme
- weitere

Bei elektromechanischen Tracking Systemen werden bewegliche Vorrichtungen an einer Person oder einem Objekt befestigt, um die bestimmten Bewegungen zu messen. Diese Vorrichtungen bestehen aus Sensoren (Drehe und Schiebewiederstände, Inkrementgeber, Flexsensoren). Diese ermöglichen eine schnelle und präzise Messungen. Die bekannteste Firma für elektromechanisches Motion Tracking auf der Markt ist „Metamotion“ [5]. Bei elektromechanischen Motion Tracking werden keine Kameras benutzt. Die Bewegungen werden direkt an dem Körper der Person oder des Objektes gemessen. An dem Objekt oder der Person werden auch Sender angebracht, die kabellos die Daten zu dem Computer des Tracking System übertragen.

Vorteile dieser Technik:

- sehr genaues und schnelles Verfahren

[5] http://www.metamotion.com/motion-capture/electro-mechanical-motion-capture.htm

- großer Aktionsradius

Nachteile dieser Technik:

- eingeschränkte Bewegungsfreiheit des Anwenders

Bei Akustischen Tracking Systemen wird mit Ultraschall gearbeitet. Es wird mit mehreren bestimmten Tönen gearbeitet um die Koordinaten des Senders zu berechnen. Der Empfänger besteht aus mehreren Mikrophonen. Diese werden kreisförmig angeordnet. Auf diese Weise kann der Sender leicht lokalisiert werden. Um jeder Sender genau zu lokalisieren muss dieser einen bestimmten Ton benutzen. Dieser Ton muss dem Empfänger bekannt sein.

Vorteile dieser Technik:

- große Bewegungsfreiheit des Anwenders
- sehr preiswert

Nachteile dieser Technik:

- leicht durch andere akustische Signale störbar
- geringere Reichweite
- geringere Genauigkeit

Bei optischen Tracking Systemen existieren zwei Arten des Verfahrens, aktive und passive. Bei aktiven Verfahren sind auch Kameras an den Objekten angebracht. Bei Passiven an den Objekten sind reflektierende Kügelchen Angebracht. Die Kameras verfolgen anhand bestimmten Figuren, die aus aus reflektierende Kügelchen gebaut sind, die Bewegung den Objekten.
Vorteile dieser Technik:

- Große Bewegungsfreiheit des Anwenders
- sehr breite Einsatzmöglichkeiten
- sehr genaues Messen
- Echtzeit, schnell

Nachteile diese Technik:

- direkte Sichtverbindung zwischen Kameras und Objekte ist nötig.
- Geringere Reichweite

Optische Tracking Systeme werden in der vorliegenden Diplomarbeit genutzt. Das Tracking-System von der Firma ART. In dem Kapitel „Einführung“ ist in der Abbildung 1.4 das ART Gerüst mit den vier Kameras zu sehen. Diese Gerüst wird benutzt um die optische Verfolgung zu realisieren. Unter optischer Bewegungsverfolgung ist die Detektion eines oder mehrerer Objekte in einem Tracking Raum zu verstehen. Um einen Punkt in einem 3D Raum zu detektieren müssen die 3D Koordinaten dieses Objektes von den Weltkoordinatensystem in das Kamerakoordinatensystem transformiert werden. Dafür werden zwei Kameras benötigt, die gleichzeitig das Objekt erfassen. Jede Kamera hat ein 2D Bild auf dem das Objekt abgebildet ist. In jeder Projektion der beiden Kameras muss das selbe Objekt gefunden werden. Das geschieht mittels „Epipolar Geometrie“ , (Meisel, 2009) , (Richard Hartley, 2003). In einem einfachen Fall wenn beide Kameras parallel ausgerichtet. Dadurch liegen die „korrespondierende Bildpunkte in der gleichen Bildzeile“ (Meisel, 2009). Wenn die beide Kameras nicht parallel ausgerichtet sind dann muss erst beide Kameras kalibriert werden. Dann ist die Position der Kamera Koordinate zu Welt Koordinate bekannt. So ist auch bekannt wie die Kameras zueinander ausgerichtet sind. Dann ist es einfacher die „Epipolarlinie“ zu finden. Wenn das zu trackenden Objekt in beiden Kamera Projektionen bekannt ist, kann in jeder Projektion eine Linie durch das Objekt und den Ursprung des Kamera Koordinatensystem gelegt werden. Die Gerade muss als eine Linie in der Ebene aufgestellt werden.

Gerade in Ebene:

$$a * x + b * y + c = 0 \tag{2.1}$$

Wenn die Projektion des Objektes einer Kamera mit der Gerade (siehe Formel 2.1) betrachtet wird, ist zu sehen, dass dieses Objekt irgendwo auf dieser Gerade liegt. Um das Objekt in der Ebene eindeutig zu identifizieren, muss den Schnittpunkt beider Gerade berechnet werden (Meisel, 2009), (Richard Hartley, 2003). Wenn die Koeffizienten beider Geraden als Vektoren aufgeschrieben werden, kann der Schnittpunkt beide Geraden mittels dem Kreuzprodukt zwei Vektoren berechnet werden.

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} * \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} \vec{e_1} & \vec{e_2} & \vec{e_3} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{pmatrix} \tag{2.2}$$

Die Vektoren $\vec{e_1}$, $\vec{e_2}$, $\vec{e_3}$ sind die Einheitsvektoren. Diese geben nur die Orientierung an. Nachdem die Koordinaten in der Ebene berechnet wurden, kann auch die dritte Koordinate mittels der Ebenengleichungen berechnet werden, (siehe (Schwald, 2006)). Bevor diese Berechnung gemacht werden kann, muss des Objektes mittels Mustererkennungs Algorithmen

im Bild die Erkennung werden (siehe (Kölzer)). Wenn die Kameras keine grosse Auflösung besitzen, dann entsteht einen Fehler erst bei Mustererkennung und dann auch bei der Berechnung der Schnittpunkt beider Geraden. Diesen Fehler muss berücksichtigen werden. Dadurch wird eine präzise Berechnung der 3D-Koordinaten des getrackten Objektes (Schwald, 2006) ermöglicht. Im MMLab der HAW-Hamburg wird mit Infrarot Tracking Systeme der Firma ART-GmbH gearbeitet. Die folgende Abbildung zeigt wie in MMLab der HAW-Hamburg die optische Bewegungsverfolgung aussieht. Auf der Abbildung wird das Gerüst (siehe Abbildung 1.4) abgebildet an dem die vier Kameras befestigt sind. In Abbildung 2.9 ist zu sehen, wie die Kameras den fliegenden Ball und die Bewegung des Roboters verfolgen. Um ein Objekt zu identifizieren, muss dieses Objekt von mindesten zwei Kameras erfasst werden. Um das zu erreichen, werden die Kameras des ART Systems synchronisiert. In jeder Kamera ist ein Embedded UNIX eingebaut. Jede Kamera identifiziert in ihrer 2D-BV[6]-Software die Markerposition in Subpixelgenauigkeit und sendet diese für jede Messung zum Tracking-Rechner, der daraus wie beschrieben die 3D-Markerposition berechnet. Mit einem Marker ist hier der ART-Body gemeint.

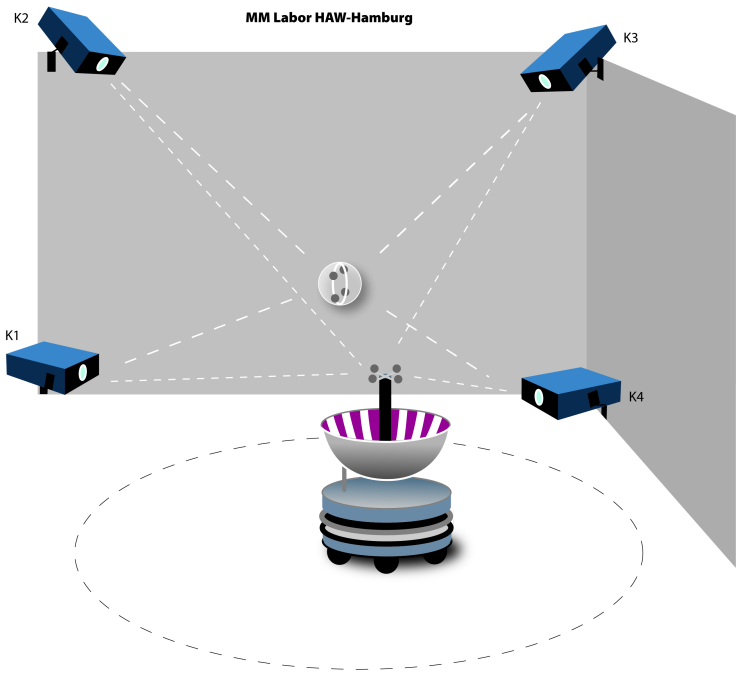


Abbildung 2.9: Roboter und Ball von ART erfasst

Das ART-System muss erst kalibriert werden, bevor es in Betrieb genommen werden kann. Es kann Raum und Body kalibriert werden. Bei der Raum- Kalibrierung wird den Ur-

[6]Bild Verarbeitung

sprung der Koordinatensystem für den virtuellen Raum festgelegt. So wie die Größe des Tracking Volumen. Bei der Body Kalibrierung werden neue Bodies dem System bekannt gegeben. Die Kalibrierung der Kameras ist sehr wichtige für präzise Arbeit des gesamten System. Durch die Kalibrierung wird auch die Ausrichtung des Kamerakoordinatensystems zu dem Weltkoordinatensystem bekannt gegeben. Ohne dieses zu wissen wird ein Realtime Tracking für den fliegenden Ball sehr schwierig. In der Abbildung 2.10 sind die Tools für die Kalibrierung des ART System abgebildet.

Abbildung 2.10: Kalibrierungs Tools ART System (ART)

Die Abbildung 2.11 zeigt die Funktionsweise des optischen Trackings bei dem ART-GmbH System. Dieses System funktioniert mittels Infrarot Licht. Die vier Kameras senden einen Infrarot Strahl mit einer Geschwindigkeit von $60\ Hz$ zu den ART Bodies. Diese bestehen aus reflektierenden Kügelchen. Der Strahl wird reflektiert und von der Kamera aufgenommen. Die Synchronisation ist auch deshalb nötig weil immer zwei Kameras gleichzeitig ein Objekt erfassen dürfen.

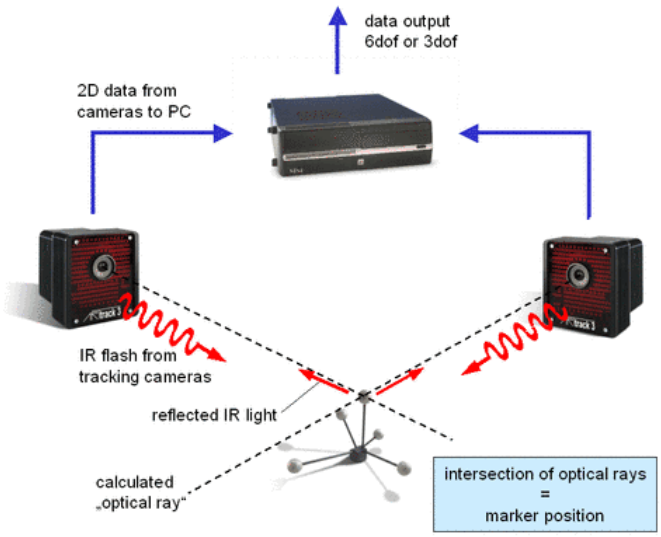


Abbildung 2.11: Optikaltracking ART System (ART)

2.4 Steuerung und Regelung der Roboterfortbewegung

Die Steuerung des Robotes unterliegt komplexen mathematischen Modellen und Algorithmen. Der Unterschied zwischen Steuerung und Regelung besteht auf der geschlossenen oder offenen Kreis (Roddeck, 2002). Bei der Steuerung wird die Ausgangsgröße nicht rückgekoppelt, der so genannte offene Regelkreis. Bei der Regelung wird die Ausgangsgröße rückgekoppelt, der so genannte geschlossene Regelkreis. Die Abbildung 2.12 zeigt wie ein Steuerungsplan aussieht. Die Abbildung 2.13 zeigt einen Regelungsplan.

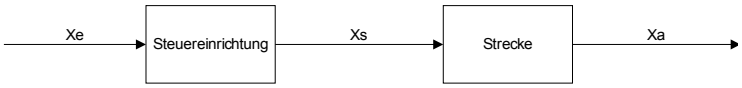


Abbildung 2.12: Steuerung, offene Regelkreis
(Roddeck, 2002)

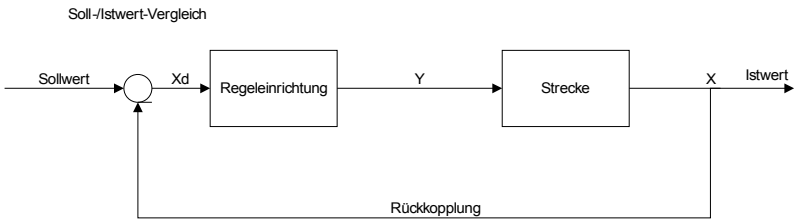


Abbildung 2.13: Regelung, geschlossene Regelkreis
(Roddeck, 2002)

Heute zu Tage benutzen die meisten Roboter für die Bewegung Elektromotorenso auch die Roboter mit Rädern. Je nach Einsatzgebiet können die Elektromotoren Servo-, Step-, Gleichstrom-, etc. Motoren sein. Das Ziel der Roboterfortbewegung ist, dass der Roboter von einer Stelle zu einer anderen sich stabil und schnell bewegt, dabei soll er seine Räder benutzen. Um den Roboter schnell und stabil zu bewegen müssen alle seine Rädern bestimmte Geschwindigkeiten haben. Dafür werden verschiedene Regler benutzt. Welcher Regler zur Regelung einer Maschine eingesetzt wird, hängt von der Regelstrecke und von dem Anwendungsgebiet selbst ab. Die Übertragungsfunktion der Regelstrecke wird aufgestellt. Durch die Sprungantwort des Gesamtsystems wird entschieden, welcher Regler/Regler-Algorithmus angewendet wird. Der einfachste Regel-Methode ist nicht immer der Beste. Der am meisten benutzten Regler in der Industrie ist der Proportional Integral Differenzial(PID) Regler (Bräunl). Bei dem Proportional Regler bleibt immer einen Fehler. Er macht nur eine Differenz zwischen dem Ausgangwert und dem Referenzwert. Wenn die Differenz negativ ist, wird der Elektromotor gebremst. Wenn die Differenz NULL ist, stoppt der Motor. Der PID Regler ist sehr geeignet zum Regelung von Elektromotoren. Die Motoren müssen je nach Fahrtrichtung unterschiedliche Fahrgeschwindigkeiten für bestimmten Zeitintervall einhalten. Deshalb müssen alle Elektromotoren des entsprechenden Roboterantriebes synchronisiert werden (Bräunl). Um die entsprechende Regelalgorithmen in einem Rechner zu implementieren, müssen die aufgestellten mathematischen Regler-Modelle in Differenzengleichung umgeschrieben werden. Auf diese Weise werden die mathematischen Modelle von Kontinuierlichen in den Diskreten Bereich umgewandelt.

2.4.1 Programmierung eines Roboter

Die Programmierung eines Roboters ist vom Aufbau und dem Einsatzgebiet des Roboters abhängig. Desweiteren hängt sie davon ab, ob der Roboter mit einem Embedded Betriebssystem oder ist nur mit eine Mikrocontroller ohne Embedded Betriebssystem ausgestattet ist. Ist auch abhängig mit welchen Programmierschnittstellen der Roboter ausgestattet ist. Verfügt der Roboter über eine API für Höhere Programmiersprache oder müssen die Instruktionen in Assembler geschrieben werden.

Ist der Roboter ein Industrieroboter mit Greifarmen oder ein Humanoid-Roboter. Der Programmieraufwand ist großer, wenn dieser in einer Maschinen(Assembler) anstatt mit eine höhere Programmiersprache programmiert wird. Die gängigen Programmiersprachen für Programmierung eines Roboters sind C,C++ und SPS. Java.

Für zeitkritische Anforderungen ist es von Vorteil wenn der Roboter über ein Realtime Betriebssystem verfügt(QNX, Wind-River, Realtime Debian Linux,...).

Um die Programme zu schreiben, zu kompilieren und auf die Zielhardware zu übertragen, muss auch ein entsprechendes IDE(Integrated Development Environment) ausgewählt werden. Diese ist von der ausgewählten Programmiersprache sowie der Zielhardware abhängig.

Programmieralternativen für den Roboter-Robotino:

Um den Robotino zu steuern ist ein Embedded PC-104 in dem Roboter verbaut. Der Roboter besitzt eine Flash Karte mit einer Größe von 256 MB oder 1 GB. Auf dieser Flash Karte ist ein Realtime Debian Linux mit Demosoftware installiert. Diese wurde in C++, C und Java geschrieben. Die Demosoftware kann auf vier Wege ausgeführt werden.

1. Direkt auf dem Roboter durch Auswahl des gewünschten Programms auf dem Roboter Display siehe Bild 1.1 und durch die Taste ENTER um das Programm starten.
2. Durch eine Verbindung per SSH (secure Shell) um sich an dem Betriebssystem des Roboters anzumelden. Dann kann das gewünschte Programm editiert werden oder mithilfe der Shell ausgeführt werden.
3. Durch Installieren der Robotino-API auf einem herkömmlichen Rechner. Das gewünschte Programm wird kompiliert und gestartet. Der externe Computer verbindet sich mit dem Robotino-Server und das Programm wird dann auf dem Robotino ausgeführt.

Es gibt noch ein andere Möglichkeit der Roboter zu Programmieren. Mit dem vom Festo mit gelieferten „RobotinoView“, dieses Programm funktioniert nach dem Prinzip(Drop and Drag). Dieses Programm ist das Programm „"LabView’ ähnlich, es bietet aber eine sehr eingeschränkte Möglichkeit für Entwicklung von Software für den Robotino.

In dieser Diplomarbeit wurde die dritte Art zum Programmieren des Robotino verwendet.

3 Technische Analyse

In diesem Kapitel werden die physikalischen und mathematischen Grundlagen behandelt, die für das weitere Vorgehen erforderlich sind. Dazu gehört die Erstellung eines mathematischen Modells, das von den Ball-Flugbahndaten eine Vorausberechnung des Ball-Landepunktes ausrechnet. Des Weiteren wird eine stabile und schnelle Anfahrtsplanung, damit der Roboter in der Lage ist, sich zu dem vorausberechneten Ball-Landepunkt zu bewegen, entworfen.

3.1 Anforderungsanalyse

Die interaktive Testumgebung mit dem Steuerrechner(handelsübliche PC mit Betriebssystem Windows XP SP3) zusammen bilden das System das nötig ist um diese Aufgabe zu bewältigen. Der Steuerrechner ist ein Notebook-T61 mit Intel-Centrino 2x2,4 Intel Pentium Dual-Core Prozessor der Firma „Lenovo“. Auf diesem Steuerrechner läuft als Host Operation System „SUSE Linux Enterprise 10.1“. Die Softwareentwicklung wird unter dem Betriebssystem Windows-XP SP3 durchgeführt. Dieses ist als Gast Betriebssystem auf eine VW-Ware Version 6.0.2 installiert.
Mit der interaktiven Testumgebung und dem Steuerrechner ist das ganze System vollständig.

Im folgenden werden die einzelnen Punkte des Systems aufgelistet:

- Optische Tracking(Beobachtung-System). Das ART System besteht aus vier Kameras und einem Tracking-Rechner. Die vier Kameras beobachten die Bodies die sich in dem Tracking-Volumen der ART befinden. Der Tracking-Rechner sendet die Koordinaten vom Ball und Roboter per UDP/IP Pakete an den Steuer-Rechner.
- Der Steuer-Rechner liest die UDP/IP Pakete ein analysiert diese und berechnet den voraussichtlichen Ball-Landepunkt. Aus der Koordinate des Ball-Landepunktes und von den aktuellen Roboter-Koordinaten werden die Roboter-Geschwindigkeit auf X- und Z-Richtung sowie Roboter-Fahrzeit berechnet. Nach Überprüfung der genannten Kriterien, werden sie an die Steuer-Funktion übergeben. Diese sendet per TCP/IP die schon berechnete Geschwindigkeit und Fahrzeit dem Roboter. Die Steuerfunktion

sorgt auch dafür, das der Roboter mit einem Steueralgorithmus immer stabil in der gewünschten Richtung fährt.

- Der Roboter empfängt die TCP/IP Pakete mit seinem Server-Programm und führt die Anweisungen aus.

Aus dem oben beschriebenen Systemzusammenhang kann folgendes Diagramm aufgestellt werden.

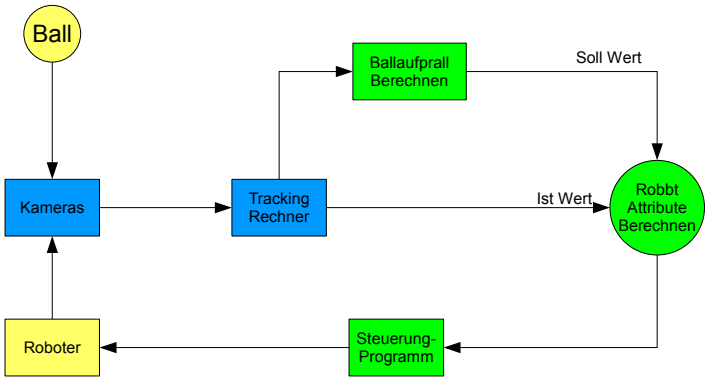


Abbildung 3.1: System Zusammenhang

3.2 Berechnungsmethoden für den Aufschlagpunkt

Bei dem schiefen Wurf handelt es sich um eine Bewegung. Ein Gegenstand, in diesem Fall ein Ball(siehe Abbildung 1), wird in einem positiven Winkel zur Erdoberfläche geworfen. Der Ball beschreibt bei idealen physikalischen Bedingungen (Windstille, keine Luftwiderstand,. . .), eine Parabel.

Erst wird allgemein das physikalische Experiment des schiefen Wurfes beschrieben, sowie die Vernachlässigung des Einflusses von einigen physikalischen Größen wie z.B Luftwiderstand, Windgeschwindigkeit etc. Ziel ist, die Beschreibung der Ballbewegung als Wurfparabel mit vernachlässigtem Luftwiderstand zu rechtfertigen.

3.2.1 Physikalische Beschreibung des schiefen Wurfs

Im ersten Schritt wird das Koordinatensystem aufgestellt.

Dieses ist in diesem Fall mit dem Koordinatensystem des ART-Systems 3.2 verbunden, denn nach der Kalibrierung des ART-Systems ist das Arbeit-Koordinatensystem bereits aufgestellt.

Bei der Kalibrierung wurde folgendes Koordinatensystem aufgestellt: X-und Z-Achse bilden die waagerechte Ebene und die Y-Achse steht rechtwinklig zu dieser Ebene.

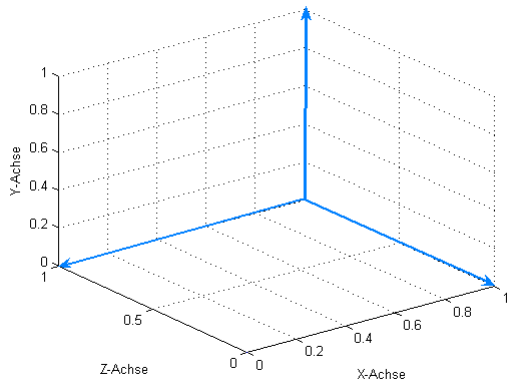


Abbildung 3.2: Ausrichtung der Koordinaten im ART System

Bei Durchführung des senkrechten Wurfs im Freien kommt die zusätzliche Störfunktion der momentanen Windkraft hinzu. Diese „Windfunktion“ ist mathematisch schwer zu berechnen, weil Richtung und Stärke des Windes sich ständig verändern. Die Windkraft muss also ständig gemessen und wenn möglich auch in einem Zeitintervall abgeschätzt werden. Die gemessene oder abgeschätzte Windkraft wird dann in die jeweilige Differenzialgleichung (DGL) (H.v.Mangoldt’s, 1961) eingesetzt. Diese DGL wird mit Hilfe der newtonschen Gesetze aufgestellt (Stroppe, 2003). Aufstellen der Differenzialgleichungen(DGL) (Manfred Reuter) in der Regelungstechnik ist eine gängige Methode um dynamischen Systems zu beschreiben.

Nach dem zweiten newtonschen Gesetz gilt:

Zweites newtonsche Gesetz

$$F_a = m * a \tag{3.1}$$

Wobei a die Beschleunigung und die zweite Ableitung des Weges nach der Zeit ist:

$$a = \ddot{x} \tag{3.2}$$

Formel 3.2 in 3.1 eingesetzt:

$$F_a = m * \ddot{x} \tag{3.3}$$

Nach dem Superpositionsprinzip ist die Summe aller Kräfte gleich einer resultierenden Kraft oder Summe aller Kräfte gleich Null:

Drittes newtonsche Gesetz Summe aller Kräfte gleich Null:

$$F_1 + F_2 + \ldots + F_n = 0 \tag{3.4}$$

Zunächst müssen alle Kräfte, die auf den Ball wirken, aufgenommen werden. Um den Ball zu werfen, muss eine Abwurfkraft F_a auf den Ball wirken. Der Luftwiderstand ist Proportional zur Geschwindigkeit (laminare Strömung) des Balls. Das ist die Luftwiderstandskraft F_l . Die Windkraft F_w ist als Störfunktion in der DGL zu betrachten. Diese Windkraft muss ständig gemessen werden, weil diese schwer mathematisch beschreibbar ist.

Schiefer Wurf unter freien Himmel:

$$F_{ax} + F_{xl} + F_{xw} = 0 \tag{3.5}$$
$$F_{ay} + F_{yl} + F_g + F_{yw} = 0 \tag{3.6}$$
$$F_{az} + F_{zl} + F_{zw} = 0 \tag{3.7}$$

Der Aufwand dieses DGL-System zu lösen ist groß, außerdem wird dieses Experiment nicht unter freiem Himmel durchgeführt, sondern in einem Labor der HAW-Hamburg. Aus diesem Grund wird die Windkraft-F_w nicht mehr betrachtet. So genügt es, diesen physikalischen Vorgang in einem zweidimensionalen Raum zu beschreiben.

(Grösse, Gewicht, Glättung) und der Luftwiderstand (Proportional zur Ballgeschwindigkeit, da es die Ballgeschwindigkeit nur eine Laminare Strömung verursacht) des Balls müsste immer noch berücksichtigt werden.

Schiefen Wurf im Geschlossenen Raum mit Luftwiderstand:

$$F_{ax} + F_{xl} = 0 \tag{3.8}$$
$$F_{ay} + F_{yl} + F_g = 0 \tag{3.9}$$

Nach Umformung der obigen Gleichungen.

Differenzial Gleichungen des Schiefen-Wurfs mit Luftwiderstand:

$$m * \ddot{x} + \vartheta * \dot{x} = 0 \tag{3.10}$$
$$m * \ddot{y} + \vartheta * \dot{y} + m * g = 0 \tag{3.11}$$

Die allgemeine Lösung einer DGL ist die Summe der homogenen und partikulären DGL:

Die allgemeinen Lösung einer DGL mit konstanten Koeffizienten:

$$H_d = H_h + H_p \tag{3.12}$$

Im Anhang A.3 sind die Differenzialgleichungen abgeleitet.

Im Anhang (A.17, A.18) ist die Variable ϑ der Luftwiderstand, der von der Form des fliegenden Objektes und des spezifischen Widerstand(Viskosität) des Mediums in diesem Fall von der Luft abhängig ist. In der obigen DGL wurde die Annahme gemacht, dass der Luftwiderstand proportional zu der Ballgeschwindigkeit ist. Bei der laminaren Strömung ist die luftwiderstandskraft proportional zur Geschwindigkeit. Laminare Strömung tritt nur dann auf wenn der Ball eine glatte Oberfläche hat und sein Geschwindigkeit in dem Medium klein ist. Z.B. wenn ein Ball mit glatter Oberfläche durch die Luft (t=20 °C) sich mit eine Geschwindigkeit von $10\ \frac{km}{h}$ bewegt. Fliegt der Ball durch die selbe Luft aber mit eine Geschwindigkeit von $60\ \frac{km}{h}$ tritt eine turbulente Strömung auf. Die turbulente Strömung die auf den Ball wirkt beeinflusst seine Flugbahn erheblich und muss im 3 dimensionalen Raum betrachtet werden. In dieser Diplomarbeit tritt nur laminaren Strömung auf, diese beeinflusst die Flugbahn des Balls aber nur minimal und kann vernachlässigt werden. Die laminare Strömung wird wie folgt ausgedrückt:

$$\vartheta = 6 * \pi * \eta * V * r. \tag{3.13}$$

Wobei η die dynamische Luftviskosität ist. Bei 20°C hat die Widerstandskraft der laminaren Strömung folgenden Betrag:

$$\vartheta = 1.5 * 10^{-}5 \frac{\mathrm{kg}}{\mathrm{m} * \mathrm{s}}. \tag{3.14}$$

Dabei ist r der Radius des Balls, der auch in der Wirklichkeit als fliegendes Objekt bei diesem Experiment verwendet wird.

Der Ball wird mit einer Geschwindigkeit von $3\ \frac{\mathrm{km}}{\mathrm{h}}$ bis $10\ \frac{\mathrm{km}}{\mathrm{h}}$ durch die Luft fliegen. So muss keine turbulente, sondern eine laminare Strömung betrachtet werden. In der Abbildung 3.3 wird die laminare Strömung an einem Ball in der Luft dargestellt.

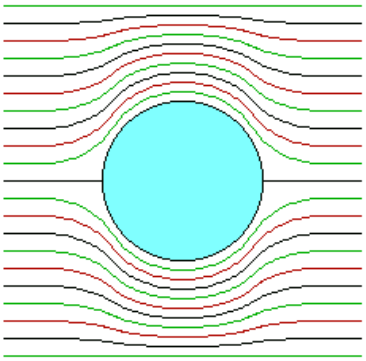


Abbildung 3.3: Laminare Streumung
(Bednarik)

Es wird angenommen werden das im Labor wo diese Experimente durchgeführt werden Windstille herrscht. Der Ball der geworfen wird wiegt 60 Gramm und hat einen Radius von 0.05 Meter. Die Oberfläche des Balls kann als ideal glatt betrachtet werden. Die Luftwiderstandskraft hat folgenden Betrag:

$$F_{rl} = 6 * \pi * \eta * V * r = 39.3 * \frac{1}{10^6} N \tag{3.15}$$

und der Betrag der Erdanziehungskraft:

$$F_g = m * g = 0.5886 N \tag{3.16}$$

Die Kraft F_{rl} ist im Vergleich zur Kraft F_g sehr klein. Das Verhältnis F_{rl} zu F_{rl} ist 0.00667. Daraus folgt, dass eine Vernachlässigung des Luftwiderstandes problemlos toleriert werden kann.

Auch wenn die Luftwiderstandskraft nicht mehr in weiteren Rechnungen berücksichtigt wird, kann das DGL-System A.17 und A.18 nicht als Ansatzlösung benutzt werden, denn der Abschusswinkel ist weiter unbekannt. Es ist sehr schwer diesen Abschusswinkel genau zu berechnen. Dieses DGL-System A.17 und A.18 ist in diesem Fall als Ansatzlösung ungeeignet.

Daher muss eine Formel gefunden werden, die den Luftwiderstand und den Abschusswinkel nicht berücksichtigt. Im ersten Schritt wird Matlab (Matlab) als Testsoftware benutzt. Es wird nach einen Algorithmus gesucht der eine Vorhersage über Landepunkt trifft. Wenn die Ergebnisse dieses in Matlab implementierten Algorithmus zufriedenstellend sind, kann dieser Algorithmus auch in einer höheren Programmiersprache so wie C++ problemlos implementiert werden.

3.2.2 Vorüberlegungen bei Lösung Model Erstellung

Das ganze System besteht aus den folgenden drei Rechner:

- ART-Tracking
- Steuerung und Berechnung
- Embedded PC-104

Der ART-Rechner schickt per UDP/IP Pakete zum Berechnungs-Rechner mit einer Geschwindigkeit von$60\ Hz$ plus der Verzögerung des ART-Systems, also kommt ca. alle $16.\bar{6}ms$ und ca. $36\ ms$ ein neues Paket an. Man bekommt somit die aktuellen Koordinaten des fliegenden Balles und des fahrenden Roboters. Diese Daten müssen danach in Echtzeit bearbeitet werden, damit der Roboter genug Zeit hat um zu den berechneten Landepunkt zu fahren um den Ball zu fangen.

Weil der Tracking-Rechner ständig neue Werte(Koordinaten des Balls und des Roboters) liefert, existieren mehr Variablen als man für eine Quadratische Parabel braucht.

Im Kapitel- 2 wird das Tracking von Objekte mit dem ART-System beschrieben. Der Ball 1 der benutzt wird dreht sich in der Luft während des Fluges und somit liefert das Tracking-System Messwerte, die nicht mehr exakt sind. Diese Umstände müssen bei weiteren Berechnungen berücksichtigt werden.

Das folgende Bild zeigt wie das Fangen des Balles durch den Roboter funktionieren soll.

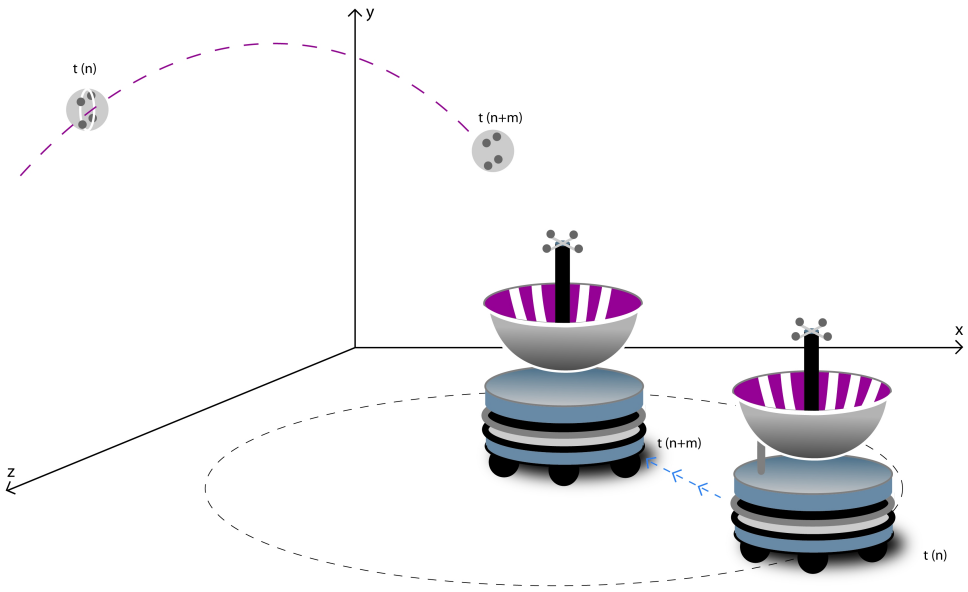


Abbildung 3.4: Wurfparabel, Roboter fängt der Ball

In der Abbildung 3.4 ist zu sehen wie der Ball in einer Parabel geworfen wird und wie der Roboter von der Position t(n) zu der Position t(n+m) sich geradlinig bewegt um den Ball mit seinem Fangkorb zu fangen. Gerade ist der kürzeste Weg zu einem Objekt zu fahren. Je früher der Landepunkt bekannt ist um so besser, da der Roboter dann mehr Zeit hat, um zu diesem Punkt zu fahren und den Ball zu fangen. Der Roboter darf aber nicht weiter als 0.6 m von der voraussichtliche Balllandepunkt entfernt sein damit er diese Stelle rechtzeitig erreicht.

3.2.3 Mathematische Modell Erstellen(Polynom-Regression)

Im folgenden wird das Verfahren der Polynom-Regression beschrieben. Betrachtet wird eine Wurfparabel ohne Luftwiderstand:

Wurfparabel Gleichung

$$y = a_2 * x^2 + a_1 * x + a_0 \tag{3.17}$$

Rein mathematisch gesehen werden nur 3 Punkte aus dieser Parabel benötigt um die Koeffizienten a_2, a_1 und a_0 der Gleichung 3.17 zu berechnen. Damit der Balllandepunkt frühst möglichst berechnet werden kann, müssen diese Koeffizienten frühzeitig berechnet werden.

Auf Grund der dargestellten Probleme bezüglich der richtigen Berechnung der Polynomkoeffizienten, ist eine geeignete Auswahl der Tracking-Daten erforderlich. Unter den Randbedingungen, dass die Gleichung schnell berechnet werden muss, da der Roboter sonst nicht genug Fahrzeit hat um den Landepunkt des Balls zu erreichen wird folgendes getan.

Es gibt eine Methode die eine Punkteschar durch ein geschätztes Polynom approximieren kann. Diese Methode ist in der Mathematik als Regression bekannt. Da nicht alle Punkte auf dem Polynom liegen, hat man immer eine Fehler. Dieser kann mit der Methode der kleinsten Quadrate von Gauß beseitigt werden. Wenn danach dieses Polynom partiell nach a_2, a_1 und a_0 differenziert wird (wenn Polynome immer stetig sind darf man die differenzieren) entstehen drei linear unabhängige Gleichungen, ein Gleichungssystem, siehe (Adam, 2006).

Die Allgemeine Beschreibung des Polynoms:

$$y = p_m(x) = \sum_{k=0}^{m}(a_k * x^k = a_0 + a_1 * x + a_2 * x^2 + \ldots + a_m * x^m) \tag{3.18}$$

Die Polynomkoeffizienten sind so zu bestimmen, dass das mittlere Fehlerquadrat minimal wird, (Renz, b).

$$\bar{D}(a_k) = \frac{1}{N} * \sum_{i=0}^{N}(a_0 + a_1 * x_i + a_2 * x_i^2 + \ldots + a_m * x_i^m - y_i)^2 = Minimum \tag{3.19}$$

Angewendet auf die Parabel bekommt man folgendes Ergebnis.

$$\bar{D}(a_k) = \frac{1}{N} * \sum_{i=0}^{N} (a_0 + a_1 * x_i + a_2 * x_i^2 - y_i)^2 = Minimum \tag{3.20}$$

In dieser Gleichung sind drei Unbekannte, deshalb muss wie vorher erwähnt die Gleichung 3.20 nach a_0, a_1, a_2 partiell differenziert werden, um drei linear unabhängige Gleichungen zu bekommen, wobei der mittlere Fehlerquadrat minimal bleiben muss.

$$A = \begin{pmatrix} \frac{1}{N} * \sum_{i=0}^{N} [2 * (a_2 * x_i^2 + a_1 * x_i + a_0 - y_i) * x_i^2 = 0] \\ \frac{1}{N} * \sum_{i=0}^{N} [2 * (a_2 * x_i^2 + a_1 * x_i + a_0 - y_i) * x_i = 0] \\ \frac{1}{N} * \sum_{i=0}^{N} [2 * (a_2 * x_i^2 + a_1 * x_i + a_0 - y_i) * 1 = 0] \end{pmatrix} \tag{3.21}$$

Nachdem mit x_i aus multipliziert wurde und die beiden Seiten der Gleichungen mit 2 dividiert werden, wird dieses Gleichungssystem in Normalform in Form einer Matrix dargestellt.

$$\begin{pmatrix} \frac{1}{N} n & \frac{1}{N} \sum_{i=0}^{N} x_i & \frac{1}{N} \sum_{i=0}^{N} x_i^2 \\ \frac{1}{N} \sum_{i=0}^{N} x_i & \frac{1}{N} \sum_{i=0}^{N} x_i^2 & \frac{1}{N} \sum_{i=0}^{N} x_i^3 \\ \frac{1}{N} \sum_{i=0}^{N} x_i^2 & \frac{1}{N} \sum_{i=0}^{N} x_i^3 & \frac{1}{N} \sum_{i=0}^{N} x_i^4 \end{pmatrix} * \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{N} \sum_{i=0}^{N} y_i \\ \frac{1}{N} \sum_{i=0}^{N} y_i * x_i \\ \frac{1}{N} \sum_{i=0}^{N} y_i * x_i^2 \end{pmatrix} \tag{3.22}$$

Um dieses Modell zu testen wurden mit Matlab (siehe Kapitel-(5)) Scripte geschrieben, die Ergebnisse sind zufriedenstellend. Es wurde eine Abweichung weniger als $1\ cm$, bei nur 30% der Sampelpunkte(Koordinatenpunkte des Balls) festgestellt. Eine Abweichung von weniger als $1\ cm$ in Vergleich des Durchmesser des Balls 1 welcher $10\ cm$ beträgt, ist tolerierbar. Es folgt ein Plot mit von Matlab aufgenommene Flugkoordinaten des Balls. Der Abwurfpunkt sowie der Aufprallpunkt sind markiert. Die Tabelle A.22 gibt die Aufgenommene Ball-Koordinaten in X,Y,Z Achsen und die ausgerechnete Koeffizienten der gesuchte quadratisches Polynom wieder.

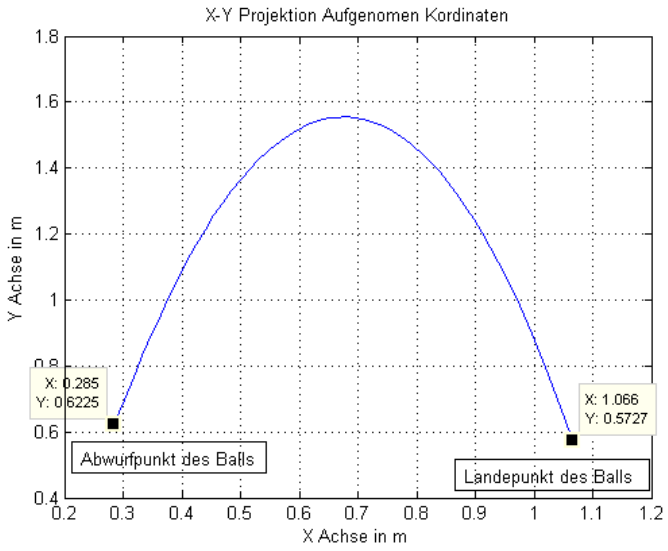


Abbildung 3.5: Koordinaten Parabel des Balls

Aufgenommenen Ball-Koordinaten in Tabellenform: Die im Abbildung 3.5 ist der Matlab-Plot den Abwurf diesen Daten-Koordinaten in der Abbildung A.22 zu sehen sind.

Der erste Wert von X und Z ist der Abwurfpunkt des Balls.Der Wert zwischen den Werten 55 und 54 ist der Landepunkt des Balls, denn der Ball wird in einer Höhe von 0.6 m gefangen. Der arithmetischer Mittelwert diese beide Punkte ist 1.059435. Jetzt folgen die Berechnete Koeffizienten der Parabelgleichung wenn alle Punkte berücksichtigt werden. Darauf folgt ein Vergleich mit den Koeffizienten dieser Gleichung bei nur 30% den aufgenommene Koordinatenpunkte des Balls.

Berechnete Koeffizienten der Parabel Gleichung bei 100% Koordinatenpunkte

$$a_2 = -6.284317, a_1 = 8.456157, a_0 = -1.887911 \tag{3.23}$$

Berechnete Koeffizienten der Parabel Gleichung bei 30% Koordinatenpunkte

$$a_2 = -6.128894, a_1 = 8.287140, a_0 = -1.845753 \tag{3.24}$$

- Bei 100% den Trackin-Daten den nach Gleichung 3.23 ist der Wert der Nullstelle(100 $\%$): 1.0630.

- Bei nur 30% den Trackin-Daten den nach Gleichung 3.24 ist der Wert der Nullstelle(30 $\%$): 1.0709.
- Der Differenz zwischen Nullstelle(100 $\%$) und Nullstelle(30 $\%$) ist weniger als $1cm$, und hat der Absolütwert: 0.0079

Mit dem Gleichungssystem 3.22 werden die Koeffizienten des quadratischen Polynoms berechnet. Mit diesen Koeffizienten werden dann die Nullstellen des Polynoms bestimmt. Es sind immer zwei Nullstellen und welche von Beiden die richtige ist wird berechnet in dem die erste x-Koordinate mit den Nullstellen vergleichen wird. Die Nullstelle die den größeren absolut Wert der Distanz zu der erste x-Koordinaten Wert hat ist die gesuchte Nullstelle. Auf der Z-Achse wird das gleiche Verfahren durchgeführt. Um den Landepunkt des Balls zu berechnen, muss erst nochmal die Parabel Betrachten werden, wenn die Parabel auf der X,Y-Projektionsebene ist, ist diese als Parabel zu sehen. Von oben auf die X,Z-Projektionsebene, ist die selbe Parabel als eine Gerade zu erkennen. Erst wird die Nullstelle berechnet, so ist die X-Komponente des Aufprall-Koordinate bekannt, dann wenn die Gerade als Projektion der Parabel ausgerechnet ist, kann mit der bekannten Nullstelle die Z-Komponente der Lande-Koordinate des Balls berechnet werden, oder die X-Komponente wenn die Nullstelle auf der Z-Achse bekannt ist.

Hier ist die mathematische Beschreibung zur Berechnen der Geraden(Projektion der Parabel auf X-Z Projektionsebene) und so die Landepunkt-Koordinaten des Balls zu berechnen.

$$\begin{pmatrix} \frac{1}{N}n & \frac{1}{N}\sum_{i=0}^{N} x_i \\ \frac{1}{N}\sum_{i=0}^{N} x_i & \frac{1}{N}\sum_{i=0}^{N} x_i^2 \end{pmatrix} * \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} \frac{1}{N}\sum_{i=0}^{N} z_i \\ \frac{1}{N}\sum_{i=0}^{N} z_i * x_i \end{pmatrix} \tag{3.25}$$

So werden die Aufprall-Koordinaten des Balls so früh wie möglich berechnet. Jetzt muss nur noch dem Roboter die entsprechenden Daten übermittelt werden, damit er dann den Ball auch tatsächlich fängt.

Nachdem die Koordinaten des Balllandepunktes berechnet sind, wird die Robotergeschwindigkeit auf der X- und Z-Achse so wie die Roboterfahrzeit berechnet. Die Roboter-Geschwindigkeit hat die Dimension $\frac{mm}{s}$ und die Roboter-Fahrzeit ms. Die Roboter-Fahrzeit bestimmt wie lange der Roboter mit der übergebene Geschwindigkeiten fahren darf. Z.B.: Der Roboter fährt 1000 ms mit einer Geschwindigkeit in der X-Richtung von 1000 $\frac{mm}{s}$ und in der Z-Richtung von 1000 $\frac{mm}{s}$. Dann ist die resultierende Geschwindigkeit $V = \sqrt{1000^2 + 1000^2} = 1.414\ \frac{m}{s}$. Fährt er mit dieser Geschwindigkeit eine Sekunde lang, dann Beträgt der zurückgelegte Weg S = 1.414 m. Um diese Geschwindigkeiten zu berechnen muss der Winkel α bekannt sein siehe Abbildung (3.6). Dieser Winkel kann mit der Arcos

Tangens Funktion berechnet werden, $dz = Rz - Bz$ und $dx = Rx - Bx$ und der Winkel wird mit $\alpha = \arctan(\frac{dz}{dx})$ berechnet. In der Abbildung (3.6) wird der Roboter mit der Buchstabe R bezeichnet und der Buchstabe B ist der vorausberechnete Balllandepunkt. Die grüne Linie zeigt wo der Roboter fahren muss. Die Geschwindigkeit V wird vorgegeben und die Geschwindigkeiten auf der X- und Z-Richtung werden mit dem Winkel-α und den trigonometrische Funktionen $\sin$ und $\cos$ berechnet. $V_x = V * \cos(\alpha)$ und $V_z = V * \sin(\alpha)$. Der Weg S der durch $S = \sqrt{dz^2 + dx^2}$ zu berechnen ist. Die Fahrzeit T ist $T = \frac{S}{V}$ und wird in ms umgewandelt.

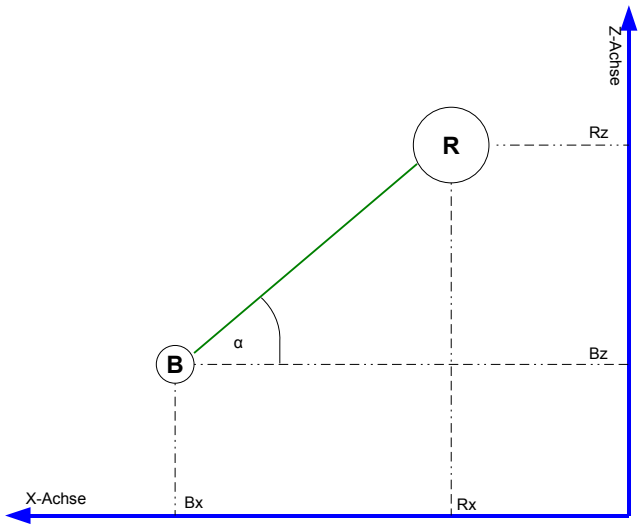


Abbildung 3.6: Der Winkel zwischen der Roboter und der vorausberechnete Balllandepunkt bestimmen

3.2.4 Berechnen der Aufprallpunkt des Balls

Hier werden zwei Methoden zur Berechnung der Ballaufprallpunkt beschrieben. Eine noch präzisere Methode als diese beiden wird als Verbesserung in Kapitel „Test und Bewertung“ beschrieben und ist im Matlab implementiert.

3.2.4.1 Erster Lösungsansatz zur Bestimmung des Balllandepunkts

Theoretische Überlegungen:
Um den Landepunkt des Balles zu berechnen muss auf eine „unverfälschte“ Projektion der Ballbahn auf eine der Projektionsebene X-Y oder Z-Y abgebildet werden. Das wird mit Hilfe einer Koordinatentransformation durchgeführt. Erst wird die Projektionsebene ausgewählt in der das Regressionsverfahren durchgeführt wird, das ist die X-Y Projektionsebene.

Der Winkel zwischen der Projektionseben X-Y und der Projektion der Parabel auf dem Boden der X-Z Projektionsebene muss gefunden werden um die Koordinatentransformationen durchzuführen.

Dafür werden die Messdaten auf den Ursprung des Koordinatensystems verlegt. Dies geschieht in dem der Mittelwert der X-Messdaten $\overline{X}$ und der Mittelwert den Z-Messdaten $\overline{Z}$ von jedem Messwert X_i und Z_i abgezogen wird. Danach wird die Korrelationsmatrix von den verschobenen Messdaten(X_i und Z_i) gebildet. Diese Korrelationsmatrix ist symmetrisch, das bedeutet das ihre Eigenwerte reale sind und ihre Eigenvektoren zu verschiedene Eigenwerte orthogonal sind siehe (Hans-Jochen, 2001). Dadurch besteht nicht die Gefahr das mit imaginären Zahlen gearbeitet werden muss.

Von dieser Korrelationsmatrix werden die Eigenwerte und die Eigenvektoren berechnet. Es muss der größte Eigenvektor-E_g zu dem größten Eigenwert berechnet werden. Diese wird zur Berechnung des Winkel zwischen der X-Achse und der Projektionsgerade der Wurfparabel benutzt. Diese E_g zeigt immer in die Richtung der Messdaten siehe (Kessler). Der Winkel zwischen der X-Achse und dem E_g wird durch das Skalarprodukt zweier Vektoren der X-Achse als Vektor $(1, 0)$ und der Eigenvektor E_g berechnet. Nachdem der Winkel berechnet ist, wird die X_i und Z_i Koordinaten um den ersten Punkt(X_0 und Z_0) mit Koordinaten X_i und Z_i gedreht.

Jetzt ist die Wurfparabel so umgedreht, dass die X-Y Projektiondebene und der Ballwurf in 2D parallel zu einander sind. Eine Berechnung der Ball-Landekoordinaten kann mit großer Genauigkeit berechnet werden. Erst werden die Nullstellen mit der schon bekannten Methode (Regression Verfahren) (Formel-3.22) berechnet, die Projektion der Parabel auf dem X-Z Projektionsebene approximiert und die Koeffizienten dieser Geraden auch mit der Methode (Regression Verfahren) (Formel-3.25) berechnet. Die Nullstelle beinhaltet die X-Koordinate und wird in die Geraden-Gleichung gesetzt und dann wird die Z-Koordinate des Ball-Landepunktes berechnet. Die berchnete X und Z Koordinate des Ball-Landepunktes werden dann mit dem selben Winkel zurück in „original Position“ um den Abwurf-Punkt(X_0, Z_0) zurück transformiert. Wenn der Winkel zwischen der Projektionsgerade des Wurfes und der X-Achse kleiner ist als 90 °, wird die hin und zurück Transformation der X und Z Ball-Bahnkoordinaten mit dem selben Winkel durchgeführt. Ist der Winkel zwischen 90 ° und 180 ° wird die hin und zurück Transformation der X und Z Ball-Bahnkoordinaten mit dem Winkel α = 180 ° - Winkelwert. Auf diese Weise wird der Trackingsvolumen nicht verlassen werden. Die Geschwindigkeit in X- und Z-Richtung sowie die Fahrzeit des Robotino wird berechnet und per WLAN übermittelt.

Die Y Koordinaten des Balles , die senkrecht zu X-Z Ebene stehen brauchen nicht transformiert oder gedreht werden. Die Matrix in der Formel 3.26 ist die Drehmatrix. Mit dieser Matrix werden alle X und Z Koordinaten um den Abwurfpunkt X_0 und Z_0 gedreht. Der Abwurfpunkt ist in der Abbildung 3.8 der Punkt P_1.

$$\begin{pmatrix} x_n \\ z_n \end{pmatrix} = \begin{pmatrix} \cos(\alpha) - \sin(\alpha) \\ \sin(\alpha)\cos(\alpha) \end{pmatrix} * \begin{pmatrix} (x_i - x_0) \\ (z_i - z_0) \end{pmatrix} + \begin{pmatrix} x_0 \\ z_0 \end{pmatrix} \tag{3.26}$$

Um die Trackingsvolumen-Grenzen beim Drehen nicht zu verlassen, müssen zwei Fälle unterschieden werden. Wenn der Winkel kleiner als 90 ° ist und wenn der Winkel größer als 90 ° und kleiner als 180 ° ist . Wenn der Winkel 0 oder 180 ° ist, ist keine Drehung und Translation nötig .

Denn wird einmal um den Uhrzeigersinn gedreht wenn der Winkel kleiner ist als 90 ° und einmal gegen Uhrzeigersinn gedreht wenn der Winkel größer ist als 90 ° ist. Wenn der Winkel größer als 90 ° und kleiner als 180 ° ist, muss mit dem Winkel gleich (180 ° - (berechnete Winkel)) gedreht werden. Auf diese Weise wird eine unverfälschte Projektion des Wurfs auf die X-Y Projektionsebene projektiert. Drehen mit der Matrix 3.26 um den Uhrzeigersinn oder gegen den Uhrzeigersinn unterscheidet sich nur um das Vorzeichen bei $\sin$ in der Nebendiagonalen der Drehematrix. So wie in der Matrix 3.26 werden die Punkte gegen den Uhrzeigersinn gedreht. Vertauscht man das Vorzeichen auf der Nebendiagonalen werden die Koordinaten-Punkte um den Uhrzeigersinn gedreht. Nachdem die Koordinaten transformiert sind wird der Landepunkt des Balles mit der Polynomregression berechnet. Mit eine Drehung um den Anfangspunkt(X_0 und Z_0) in die entgegengesetzte Richtung werden die Berechneten Lande-Koordinaten des Balles wieder in die Originalposition gebracht. Danach wird die Entfernung mit Hilfe des Pythagoras berechnet und die Roboter-Geschwindigkeiten auf der X und Z Richtung mit Hilfe von cos für X-Richtung und sin für Z-Richtung so wie die Fahrzeit des Roboter berechnet. Die optimale Geschwindigkeit wird als Konstante eingegeben. Die Fahrzeit wird durch teilen der Distanz zwischen dem Roboter und Landepunkt des Balls und die festgelegte Roboter-Geschwindigkeit(V_{max}) berechnet. Um die Geschwindigkeiten auf X und Z Richtung zu berechnen wird erst der Winkel(α) zwischen Hypotenuse(Fahr-Distanz) und der X-Kathete berechnet.

Robotegeschwindigkeit auf X Richtung:

$$V_x = cos(\alpha) * V_{max} \tag{3.27}$$

Robotegeschwindigkeit auf Z Richtung:

$$V_z = sin(\alpha) * V_{max} \tag{3.28}$$

Diese Methode, die beschrieben wurde ist für die Winkel von 0 ° bis 180 °. Die selbe Methode kann auch für die Winkel von 0 ° bis -180 ° angewendet werden. Es hängt davon ab wie der Winkel gemessen wird von 0 ° bis 180 ° und auch im negativen Bereich oder 0 ° bis 360 °.
Abbildung 3.7 zeigt wie ein Punkt auf X-Z Ebene gedreht wird. Punkt P1 wird um den Winkel α gedreht auf P2. Die Koordinaten auf der Y-Achse bleiben unverändert, so müssen nur die X und Z Koordinaten sich um einen Winkel drehen. Hier wird um den Koordinatenursprung gedreht. In der Aufgabe muss diese durch eine Translation manipuliert werden und dann um den Anfangspunkt(X_0 un Z_0) die X und Z Koordinaten des Balls gedreht werden.

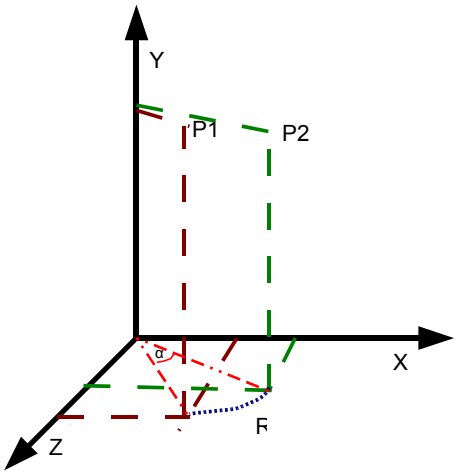


Abbildung 3.7: Drehen ein Punkt auf X-Z Ebene

In folgenden Grafiken wird gezeigt wie der Winkel α zwischen X-Achse und die Projektionsgerade berechnet wird. In der Abbildung 3.8 ist ein Wurf-Fall so wie oben beschrieben graphisch dargestellt. Nur hier wird die Drehbewegung um den Koordinatenursprung durchgeführt. Um die Drehbewegung um den ersten Messpunkt durchzuführen, muss eine Translation durchgeführt werden.

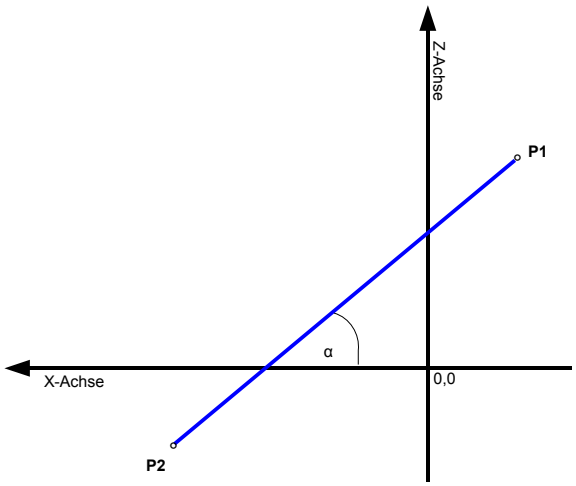


Abbildung 3.8: Wurfprojektion auf X-Z Projektionsebene

In Abbildung 3.9 sind die auf den Ursprung versetzten X_i und Z_i Koordinaten dargestellt. Danach, so wie oben beschrieben, wird der Winkel zwischen der X-Achse und E_g berechnet. Der Eigenvektor E_g ist hier auf der Abbildung 3.9 nicht dargestellt. Dieser Eigenvektor wird in dem Verbesserungsverfahren, das später vorgelegt wird und in der Matlab Simulation implementiert und simuliert, gezeigt.

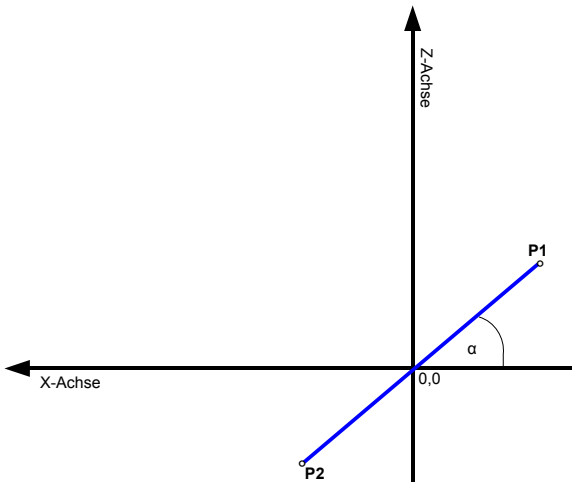


Abbildung 3.9: Verlegte Wurfprojektion auf X-Z Projektionsebene

In folgenden Abbildungen wird die Drehung um den ersten Messpunkt P(X(0),Z(0)) einmal mit Winkel kleiner als 90 °:

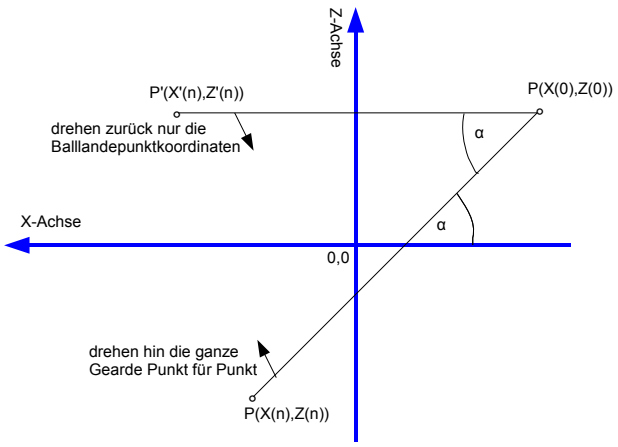


Abbildung 3.10: Dreheung wenn der Winkel kleiner als 90 ° ist

In Abbildung 3.10,wird den Gerade[1] zwischen die Punkte P(X(0),Z(0)) und P(X(n),Z(n)), gezeigt wie diese um den Abwurfpunkt um der Winkle α gedreht wird.
Diese Gerade muss mit dem Winkel(α), der wie oben beschrieben von dem Skalarprodukt zwei Vektoren ausrechnen und im Uhrzeigesinn um den Anfangspunkt-P(X(0),Z(0)) die selbe Gerade drehen. Nach der Dreheung werden die Koordinaten des Ball-Landepunktes berechnet. Die Ball-Landekoordinaten müssen danach wieder in die „original“ Position gebracht werden. Dies gescheit in dem die Ball-Landekoordinaten als Vektor(transponiert) mit der Drehmatrix 3.26 aber in entgegengesetzte Richtung (der Vorzeichen in Nebendiagonal der Drehmatrix negieren) multiplizieren. So werden jetzt die wahren Ball-Landepunktkoordinaten berechnet und weiter damit gearbeitet um den Roboter zu steuern.

Und einmal die Drehung um den ersten Punkt P(X(0),Z(0)) mit dem Winkel größer als 90 ° und kleiner als 180 °:

[1] die Projektion der Parabel auf X-Z Projektionsebene

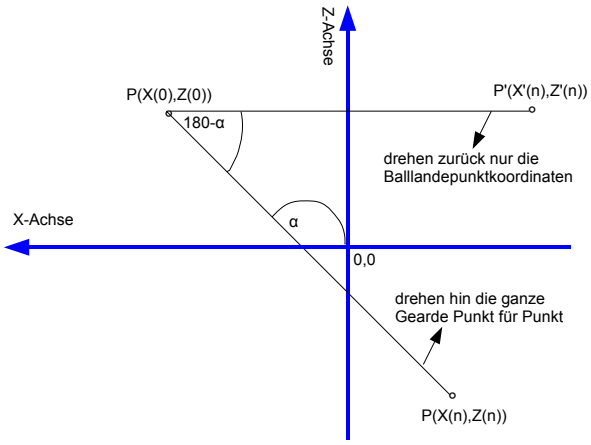


Abbildung 3.11: Dreheung wenn der Winkel grösser als 90 ° und kleiner als 180 ° ist

Im Abbildung 3.11 ist gezeigt wie die Punkte der Gerade zwischen die Punkten P(X(0),Z(0)) und P(X(n),Z(n)) um den Abwurfpunkt P(X(0),Z(0)) gedreht werden wenn der Winkel(α) größer ist als 90 ° und kleiner ist als 180 °.
Es muss ein neuer Winkel(β) ausgerechnet werden, der die Differenz zwischen 180-(α) ist. Auf diese Weise bleibt die Transformation immer in den Grenzen des Trackingsvolumens des ART-Systems. Wenn die Transformation durchgeführt ist, werden die Ball-Landekoordinaten berechnet. Der Vektor der Ball-Landekoordinate wird transponiert und mit der Drehmatrix 3.26 multipliziert damit diese die Koordinaten des Balles wieder in den richtigen Stelle zurück transformiert. Auch in diesem Fall werden die wahren Ball-Landekoordinaten berechnet, des weiteren werden sie gebraucht um der Roboter zu steuern.

Auf diese Weise werden die Transformationen der aktuellen Roboterkoordinaten auf der X-Z Projektionsebene gespart, die auch um den ersten Punkt P(X(0),Z(0)) gedreht ist. Es werden einmal alle Ballkordinaten Transformiert, dann die Berechnung der Ball-Landekoordinaten durchgeführt und schließlich diese Ball-Landekoordinaten erst als Vektor transponiert und dann zurück transformiert. Dieser Algorithmus wird so lange durchgeführt bis der Ball von dem Roboter gefangen wird.

3.2.4.2 Zweite Lösungssatz für Landepunkt des Balles

Weil der Ball (siehe Abbildung (1)) aus zerbrechlichem Kunststoff besteht, darf dieser nicht auf den Boden aufschlagen. Um das Aufschlagen zu verhindern wird der Ball an einer Angelsehne aufgehängt. Diese Angelsehne hat allerdings den Nachteil, dass sie die Flugbahn des Balls beeinflussen kann. Die erwartete Gerade als Projektion der Wurfbahn auf X-Z

Projektionsebene trifft selten zu. So kann der Winkel der in der ersten Methode(Erste Lösungssatz für Landepunkt des Balls) gebraucht wurde hier schwer und auch selten präzise genug berechnet werden.

Aufgrund oben genannter Einschränkungen wurde zunächst eine einfache Lösung gewählt die nicht von dem Winkel zwischen der Projektionsgerade und X-Achse abhängig ist.

Wenn der Ball geworfen wird, beschreibt dieser ein Parabel auf X-Y Projektionsebene oder auf Z-Y Projektionsebene oder auf Beiden, wenn die Projektionsebene auf X-Z Ebene(Fußboden) ein Winkel mit X Achse um ca. $\frac{1}{4}\pi$ ist. Das folgende Bild zeigt die Projektionen zweier Wurfparabeln auf der X-Z Projektionsebene.

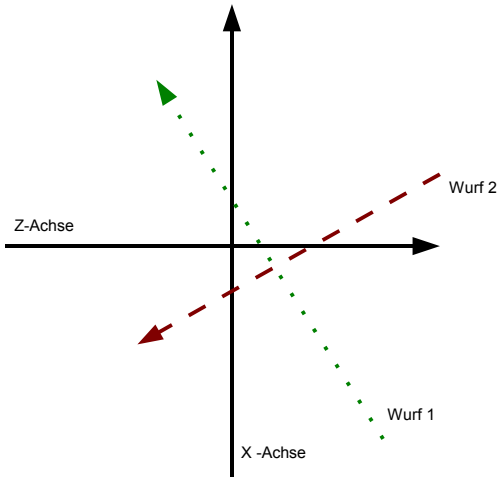


Abbildung 3.12: Parabel Projektion auf Z-X Ebene
referenzieren

Die gestrichelte Wurfparabel kann auf der X-Y Projektionsebene projektieren und ohne nennenswerte Abweichungen die Nullstellen dieser Parabel bestimmen, weil der Winkel zwischen dieser Projektion und X Achse kleiner als $\frac{1}{4}\pi$ ist. Die andere gepunktete Wurfparabel kann man aber nicht mehr auf X-Y Projektionsebene projektieren, denn der Winkel zwischen dieser Projektion und X Achse ist größer als $\frac{1}{4}\pi$. Diese Wurfparabel muss man auf der Z-Y Projektionsebene projektieren um die richtigen Nullstellen zu berechnen.

Es wird in einer Projektionsebene(X-Y oder Z-Y) das Regression-Verfahren durchgeführt. Danach auf Grund des Ergebnises dieses Verfahren werden die Roboter-Steuerkomponente, die Robotergeschwindigkeit auf X- und Z-Richtung und die Roboterfahrzeit berechnet. Entspricht eine von dieser Roboter-Steuerkomponente die vorgegebenen normierten Werte wird die Projektionsebene gewechselt. Auf diese Weise wird die richtige Projektionsebene gefunden.

Mit dieser Methode, wenn der Winkel zwischen Parabelprojektion auf X-Z Projektionsebene(Fußboden) und X-Achse oder Z-Achse größer ist als $\frac{1}{6}\pi$, sind die Nullstellen nicht so genau zu berechnen wie wenn der Winkel kleiner ist als $\frac{1}{12}\pi$. Aber dieser Fehler ist etwa um 2 bis 3 cm im Vergleich mit der Größe des Balles Diameter 0.1 m und der Größe des Fangkorbs 0.32 m,der auf dem Roboter installiert ist, ist zu vernachlässigen. Der Fangkorb ist ca. 3 mal größer als der geworfene Ball und kann so auch größere Abweichungen bei den Nullstellen kompensieren.

3.3 Steuerung und Regelung des Roboters

In Kapitel- 2 wurde der verwendete Roboter beschrieben. An dieser Stelle werden dessen Fahreigenschaften, wie z.B. Beschleunigung, Stabilität der Fahrtrichtung, Beschleunigung und Bremsweg analysiert. Diese Überlegungen bilden die Grundlage für die Wahl und Entwurf der am besten geeigneten Form der Steuerung.

3.3.1 Fahrverhalten des Omnidrive-Systems vom Robotino

Der Robotino hat einen Omnidrive-Antrieb, der aus drei sternförmig angeordneten Gleichstrommotoren besteht. Dies steigert seine Rangiermöglichkeiten enorm. Wie der Name schon sagt; „omnidrive“ bedeutet „in jede Richtung fahrbar“ und zwar ohne vorherige Rotation. Die Rotationsbewegung ist auch möglich und problemlos mit diesem Antrieb durchzuführen, in dieser Aufgabe wird er aber nicht benötigt.

Der Roboter wird sich im Bewegung gesetzt indem man ihm eine Geschwindigkeit in X- und eine in Z-Richtung per WLAN übergibt. Er fährt sofort und direkt in die gewünschte Richtung ohne vorher eine Drehbewegung durchzuführen. Er kann während der Fahrt neue Werte für X- und Z- Richtung empfangen und so eine gleichmäßige Bewegung ausführen. Das Omnidrive-System wurde im Kapitel 2 kurz vorgestellt.

3.3.2 Vorüberlegungen zum Steuerungsmodell des Roboters

Der Roboter muss so schnell wie möglich die voraussichtliche Aufprallposition des Balles erreichen, damit er den Ball noch in der Luft fängt. Für typische Würfe innerhalb des Tracking-Volumens von ca. 1.5 m Höhe ergeben sich die Zeiten von ca. 700-800 ms für die Wurfdauer. Da der Roboter als physikalisches System betrachtet, eine nicht zu vernachlässigende Masse hat, müssen entsprechend auch die Beschleunigungszeit, den Schlupf und den Bremsweg berücksichtigt werden. Von der Geschwindigkeit hängt auch die maximale

Entfernung zwischen dem Ausgangspunkt des Roboters und dem voraussichtlichen Landepunkt des Balles ab.

Im Wesentlichen geht es hier um eine kontinuierliche Nachführung des Roboters auf seinen Weg zum Aufprallpunkt. Der minimale Zeitabstand, mit dem er neue Daten per WLAN bekommt liegt bei $60\ Hz$. Dies bedeutet, dass die Zeit, die das Steuerungsprogramm zur Verfügung hat um die neuen Daten per UDP/IP zu berechnen, auch in diesem Bereich liegt.

Die höchste Geschwindigkeit, die der Roboter sinnvoll nutzen kann, hängt von vielen praktisch schwer modellierbaren Faktoren ab, wie Schlupf der Räder, Akkuladung etc. Aus diesem Grunde wurde die optimale Geschwindigkeit, d.h. diese, bei der die Bewegung des Roboters stabil ist, in einer Reihe von Experimenten bestimmt.

Unter dem Begriff stabil versteht man, dass der Roboter nach Vorgabe der Geschwindigkeiten in X- und Y Richtung, eine noch tolerierbare Abweichung (+/- 5 cm)von der theoretischen Spur aufweist.

Um diese Geschwindigkeit zu ermitteln wurde der Roboter in mehreren Versuchen mit verschiedenen Geschwindigkeiten auf beiden Achsen angetrieben. Diese wurden so ausgewählt, dass der Roboter immer auf einer Gerade, die um 45 Grad zur X-Achse geneigt ist, fahren muss. Die entstehenden Abweichungen von dieser Gerade werden erfasst, analysiert und der durchschnittliche Fehler ermittelt.

Es wurden zwei Reihen von Experimenten durchgeführt:

- Einmalige Vorgabe der Geschwindigkeiten am Anfang der Bewegung
- Mehrmalige Übertragung der Geschwindigkeiten während der Bewegung.

Die Ergebnisse der Experimente sind in Anhang 1 und 2 dargestellt. Nach Analyse der Daten ist festzustellen, dass die Genauigkeit und die Stabilität der Bewegung mit einmaliger Vorgabe der Geschwindigkeit bei weitem nicht ausreichen sind. (Siehe Appendix 1)

Aus diesem Grund wurde nach einer genaueren Art der Robotersteuerung gesucht. Als wesentlich besser geeignet erwies sich die Variante, bei der die Geschwindigkeiten mehrmals in regelmäßigen Zeitabständen während der Fahrt übertragen werden. Das führt zu einer stabilen Bewegung des Roboters bei Geschwindigkeiten von ca. 700 mm/sec. Als oberste Grenze der Stabilität erweisen sich 800 mm/sec. (Siehe Appendix 2)

Der Roboter hat ein eigenes Koordinatensystem, welches relativ zu dem ART-Koordinatensystem (Basis-Koordinatensystem) ist. Zu Beginn muss der Roboter in eine bestimmte Lage relativ zum Basiskoordinatensystem aufgestellt werden, da die Z-Achse des Roboterkoordinatensystems in die negative Richtung der Z-Achse des Basis-Koordinatensystems zeigt. Aus diesem Grund muss der Roboter immer in einer bestimmten Lage zum Basis-Koordinatensystem gehalten werden. Da die Ausrichtung der Z-Achse des Roboters in die entgegengesetzte Richtung zeigt, muss der Betrag der Z-Koordinate negiert werden, um den Roboter in die gewünschte Richtung fahren zu lassen.

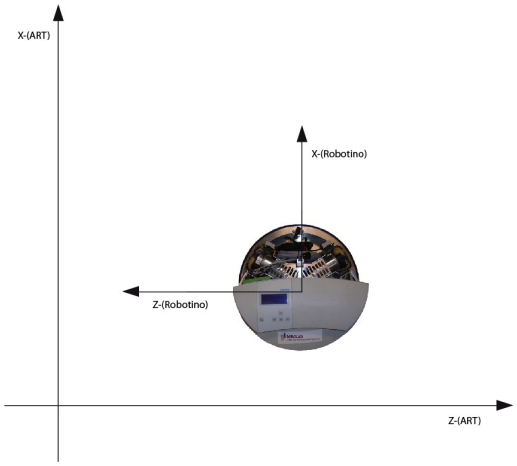


Abbildung 3.13: Robotino relative Koordinaten

3.3.3 Erstellen des Steuerungsmodells

Die Kameras des ART-Tracking-Systems überwachen den Roboter mit $60\ Hz$(siehe Abbildung 2.9). Dies bedeutet, dass in Zeitabständen von ca. $16.6\ ms$ Datenpakete vom Steuerrechner verarbeitet werden müssen. Aus diesen Daten werden die Richtung, die Geschwindigkeit(auf X-Achse und auf Z-Achse) und die Zeit ,die der Roboter fahren muss, errechnet. Diese Parameter werden dann per WLAN an den Roboter übertragen, um den vermuteten Landepunkt des Balles zu erreichen.

Wenn die vom ART-System erfasste Position des Roboters von der theoretisch errechneten abweicht, muss eine Korrektur vorgenommen werden und ein neues Paar von Geschwindigkeiten(auf X-Achse und Z-Achse) so wie neue Fahrzeit berechnet werden. Diese Geschwindigkeiten und Fahrzeit werden in der nächsten Zeiteinheit von $\frac{16}{6}\ ms$ per WLAN zum Robotino übertragen. Diese Korrektur geschieht während der Roboter fährt. Es wird nicht überprüft ob der Roboter von der Position abweicht, die neuen Daten werden ununterbrochen berechnet und ihm während der Fahrt übermittelt.

Der Weg den der Roboter auf der X- und Z-Achse fährt, kann mit folgenden Ausdruck 3.29 beschrieben werden.

Berechnete Koeffizienten der Parabel Gleichung bei 100% Koordinatenpunkte

$$x(t+1) = x(t) + dx \tag{3.29}$$
$$z(t+1) = z(t) + dz \tag{3.30}$$

In der Formel 3.29 ist zu sehen, dass wenn dx und dz zu groß sind, kann der Roboter größere Abweichungen beim Fahren verursachen. Ist der Roboter dann zu weit abgewichen, ist eine Korrektur aus Zeitmangel nicht mehr möglich. Aus diesem Grund müssen die dx und dz optimal klein sein.

Der Roboter fährt immer den kleineren der Weg „S“, $S = \sqrt{dx^2 + dz^2}$. Wenn der Roboter eine Abweichung beim Fahren gemacht hat, wird seine Position von den Sensoren erfasst. Bei Berechnung der neuen Geschwindigkeiten auf X und auf Z Richtung sowie Fahrzeit, wird diese Abweichung korrigiert. Auf diese Weise ist es Möglich den Roboter sicher auf die Ball-Landeposition zu steuern.

In der folgenden Abbildung 3.14 ist gezeigt, wie der Roboter zu dem Ball-Landepunkt gesteuert wird. Das ist ein möglicher Fahrt der Roboter zu dem Ball-Landepunkt.

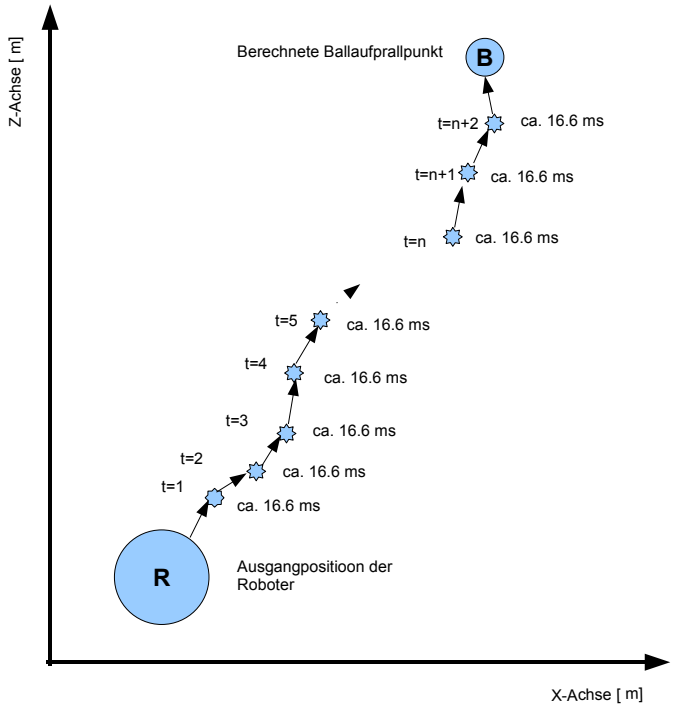


Abbildung 3.14: Steuerung der Roboter

Im Kapitel Test und Bewertung sind die zugehörige Diagramme zu finden 5.2.

Der Roboter fährt in eine Richtung die vorher ihm von dem übertragenen Geschwindigkeiten bestimmt wurde. Wenn eine neue Geschwindigkeit ihm übermittelt wird und diese mit der Geschwindigkeit die eine Zeiteinheit vorher übermittelt wurde übereinstimmt, fährt der Roboter auf der alten Richtung weiter, sonst korrigiert er seine Richtung.

Dieser Algorithmus ist einfach zu implementieren, erfordert aber eine aufwändige Parametrierung. Diese Parameter wurden durch praktische Tests - die mit dem Roboter durchgeführt wurden - und durch das entwickelte theoretische Steuerungsmodell immer weiter verbessert, so dass am Ende ein zufriedenstellendes Ergebnisse erzielt wurde. Die Konstante die die Zeit für diesen Algorithmus in kleine Portionen zerteilt ist nur experimentell zu bestimmen.

Verweis auf Kapitel „Anhang“ A.23 wo Test mit den Geschwindigkeiten, Fahrzeiten und Systemzeiten zu finden ist.

4 Design und Implementation

Um ein gutes Design und davon auch eine gute Implementierung zu ermöglichen, wird erst einen Regelung-Technischer Plan entworfen. Auf Basis dieses regelungstechnischen Plans werden dann die benötigte Strukturen und Interaktionen entworfen. Bei der Implementation auf eine höhere Programmiersprache wird dann der Realtime Aspekt berücksichtigt.

4.1 Regelungstechnisches Modell

In der Praxis wird oft ein Regelkreis des Systems erstellt. Hier handelt es sich um ein dynamisches System [1]. Da die Ausgangroße des Systems in dieser Aufgabe die Ball-Landekoordinaten sind und immer neu berechnet werden und diese nicht nur von der Eingangsgröße der aktuellen Ballkoordinaten abhängig sind, sondern auch von den vorherigen Ballkoordinaten, wird dieses System als dynamisch bezeichnet.

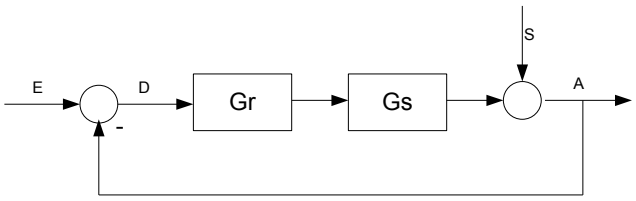


Abbildung 4.1: Allgemeines Modell eines Regelkreises

Um einen Regelkreis für ein System zu entwerfen, muss erstmal in Erfahrung gebracht werden, wie ein Regelkreis aufgebaut ist und welche die Komponenten eines Regelkreises sind. Die grundlegende Elemente eines Regelkreises sind Regler-Gr und die Regelstrecke-Gs (siehe Abbildung 4.1), dazu gehören ein oder mehrere Eingangssignale, die Differenz zwischen Eingangssignal(Soll-Wert) und des Ausgangssignals (Ist-Wert), sowie ein oder

[1] Wenn die Ausgangsgröße zum Zeitpunkt t=t1 nicht nur von dem momentanen Wert der Eingangsgröße zu Zeitpunkt t=t1 abhängt, sondern auch von der Vorgeschichte der Eingangsgröße im Intervall $0 \leq t \leq t1$, so wird das System als dynamisch bezeichnet (Skript Regelungstechnik-1, Professor Suhl HAW-Hamburg)

mehrere Störgrößen. Ziel eines Regelkreises ist es eine auf Grund einer Störung(S), die Ausgangsgröße(A) und die Eingangsgröße(A), die Regeldifferenz(D9D zwischen Eingangsgröße-E und der Ausgangsgröße-A möglich schnell zu beseitigen oder möglichst klein zu halten (siehe (Braun, 2005)).

Auf Grund der Regelkreis in Abbildung 4.1 wurde auch der Regelkreis für das System im diese Diplomarbeit siehe Abbildung 4.2.
In Abbildung 4.2 ist der Soll-Wert der Ball-Landepunkt, das ist auch gleichzeitig die Zielposition des Roboters. Die Differenz zwischen Soll- und Ist-Wert soll immer kleiner werden damit der Algorithmus terminiert, das heißt den Roboter hat der Ball gefangen. Die Störgröße-S ist unbekannt. Diese Größen sind im Experiment schwer zu messen.
Dieser Regelkreis soll als Designe der Software für dieses Projekt dienen. Dieses Regelkreis-Modell besteht aus drei Komponenten:

- Sensoren(ART-Kameras) mit dem ART Tracking-Rechner mit Blau bezeichnet
- Regler(Steuerungs im Steuer-Rechner) Berechnung und Steuerung Programmen mit grün bezeichnet
- Regelstrecke(Roboter) und Sollwertvorgabe(fliegende Ball) mit Gelb bezeichnet

Beschreibung: Die aktuelle Position des Roboters und des Balls wird von dem ART-System erfasst. Das ART ist der Sensor in diesem System. Die Daten bekommt der Steuerrechner. Dieser errechnet die Ball-Landepunkt-Koordinaten voraus. Das Ergebnis ist der Soll-Wert. Die aktuelle Position des Roboters ist der Ist-Wert. Von dem Ist- und dem Soll-Wert werden die Steuerparameter des Roboters berechnet. Die Steuerparameter werden dem Roboter in einem Steueralgorithmus übertragen. Der Roboter führt die Instruktionen aus.

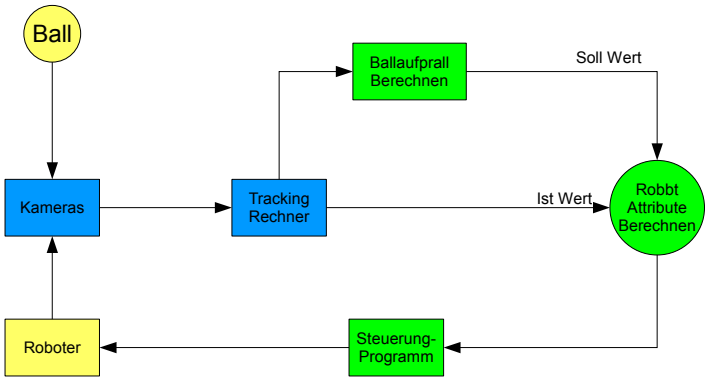


Abbildung 4.2: Regelung-Technisches Modell

Der nächste Punkt ist von dem Regelkreis 4.2 die Struktur der Software zu entwerfen. Wie

viele Klassen werden benötigt, wie sollen implementiert werden und wie soll dann die Interaktion zwischen den Instanzen dieser Klassen funktionieren.

4.2 Daten und Klassen Struktur

In diesem Kapitel werden die Klassenstrukturen entworfen. Das Struktur des Systems besteht aus drei Teilen.

- Tracking
- Berechnung und Steuerung
- Ausführung

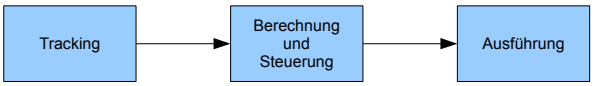


Abbildung 4.3: Strucktur das gesamte System

Das Design der Software wurde stufenweise durchgeführt.

- Erst wurde die Kommunikation mit dem Roboter hergestellt und kleine Programmen erstellt um sein Fahrverhalten zu überprüfen.
- Die Kommunikation mit dem ART-Rechner erstellt
- Das Programm zum Speichern der Ballkoordinaten.
- Das Programm das die Ball-Landekoordinaten ausrechnet.
- Das Programm das den Roboter steuert.
- Das Programm zur Steuerung des Roboters, wird als nebenläufig umgewandelt.
- Test Routinen in Klassen um dessen Verhalten zu analysieren.
- Anhand der Testroutinen wird das ganze Programm analysiert und geändert.

Die Reihenfolge des Programmablaufes ist auf dem folgenden Muster aufgebaut siehe Abbildung 4.4. Die Phasen sind Nummeriert und werden von Erläuterungen gefolgt.

1. Programm wird gestartet.
2. Initialisierung der Variablen

3. Kommunikation mit Roboter und dem ART-Rechner aufbauen.
4. Berechnung starten, aktuelle Koordinaten des Balls und des Roboters einlesen und speichern.
5. Die Roboter-Steuerung wird nur ein mal gestartet.
6. Berechnen von Ball-Landekoordinaten, berechnen der Roboter-Geschwindigkeit und Roboter-Fahrzeit.
7. Berechnung beenden.
8. Programm beenden.

Die Berechnung Punkt-4 wird erst dann gestartet wenn der Ball bei dem Flug eine vorher festgelegte Höhe überschreitet. Punkt-4 und Punkt-6 laufen ununterbrochen in einer Schleife bis das Programm beendet ist. In Punkt-5 wird die Roboter-Steuerung nebenläufig gestartet. Die Roboter-Steuerung muss ein nebenläufiger Prozess sein, weil der Roboter permanent auf Abweichungen reagieren muss.

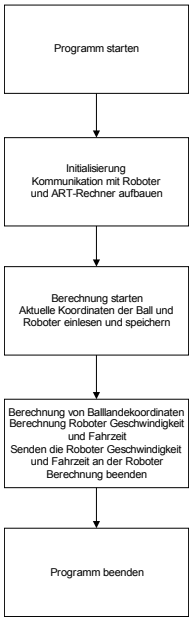


Abbildung 4.4: Programm Ablauf

In Abbildung 4.5 ist das UML Klassendiagramm zu sehen. Die in Gelb bezeichneten Klassen sind die Klassen, die die Kommunikation mit dem ART-System aufbauen. Dieser Klassen sind von dieser Adresse übernommen (HAW-Hamburg). An diesen Klassen ist nichts verändert. Die Klasse $CTrack$ zusammen mit dem Struct $bodyStruc$ ist von einem vorigen

Projekt übernommen. Diese Klasse ist verändert und angepasst für diese Aufgabe. Die Klasse Com von der Robotino-API wird benutzt und teilweise verändert und für diese Aufgabe angepasst. In der Klasse $CTrack$ wird erst die Initialisierung der Variablen für die übrigen Klassen durchgeführt. Danach wird die Kommunikation zum Roboter hergestellt und die Initialisierung den el.Motoren sowie der Bewegungssensoren. Danach wird die Kommunikation mit dem ART-Rechner hergestellt. Die UDP-Pakete werden ausgelesen und in die Klasse $CUDPDaten$ gespeichert. In der Klasse $Parabel$ wird das quadratische Polynomregression durchgeführt und die Nullstellen dieses quadratischen Polynoms berechnet. Dafür braucht die Klasse $Parabel$ alle vorherigen samt der aktuellen Ballkoordinaten. Diese Koordinaten bekommt die Klasse $Parabel$ von der Klasse $CUDPDaten$.

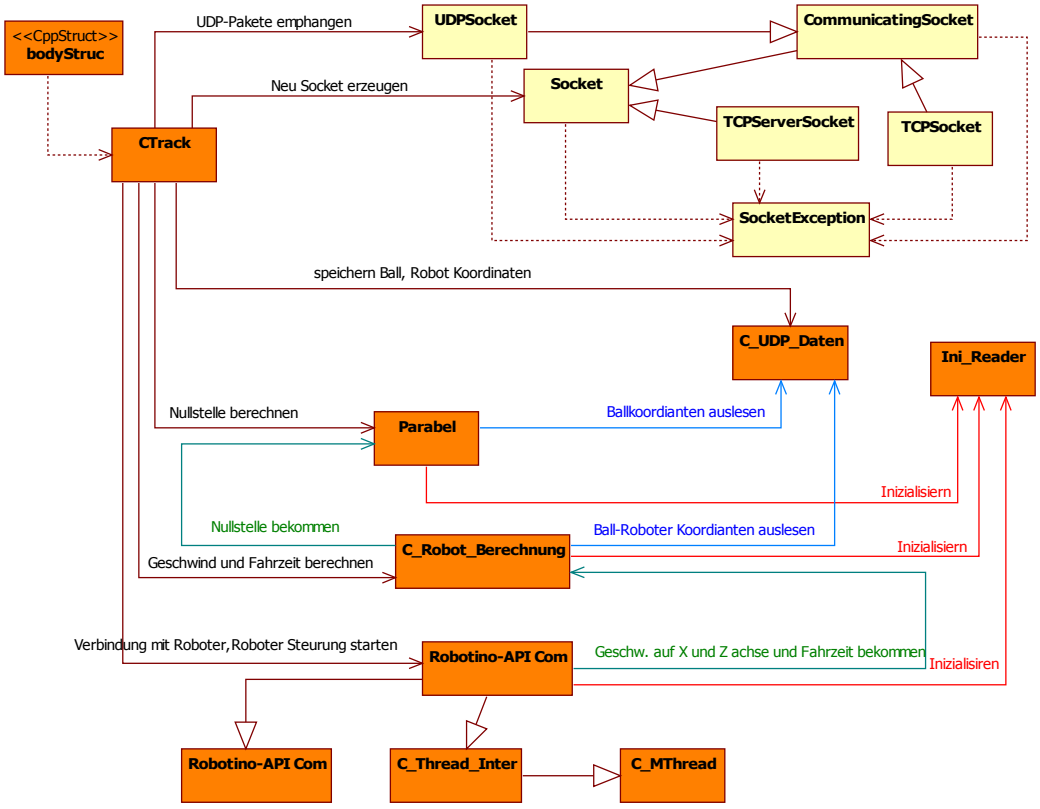


Abbildung 4.5: die Klassen in ganzes Programm

In Phase-1 wird das Programm aus der Konsole gestartet. In Phase-2 wird die Initialisierung der Variablen durchgeführt. In Phase-3 wird die Kommunikation mit Roboter und ART-Rechner aufgebaut. Phase-4 wird erst gestartet wenn der Ball eine bestimmte Höhe überschreitet, danach läuft diese ununterbrochen. In dieser Phase wird nur einmal die Robotersteuerung als nebenläufiges Programm gestartet und läuft dann im Hintergrund weiter.

Dies geschieht, da vorher keine vernünftigen Werte zur Robotersteuerung berechnet wurden. Wenn der Ball eine bestimmte Höhe unterschreitet, wird die Berechnung beendet.

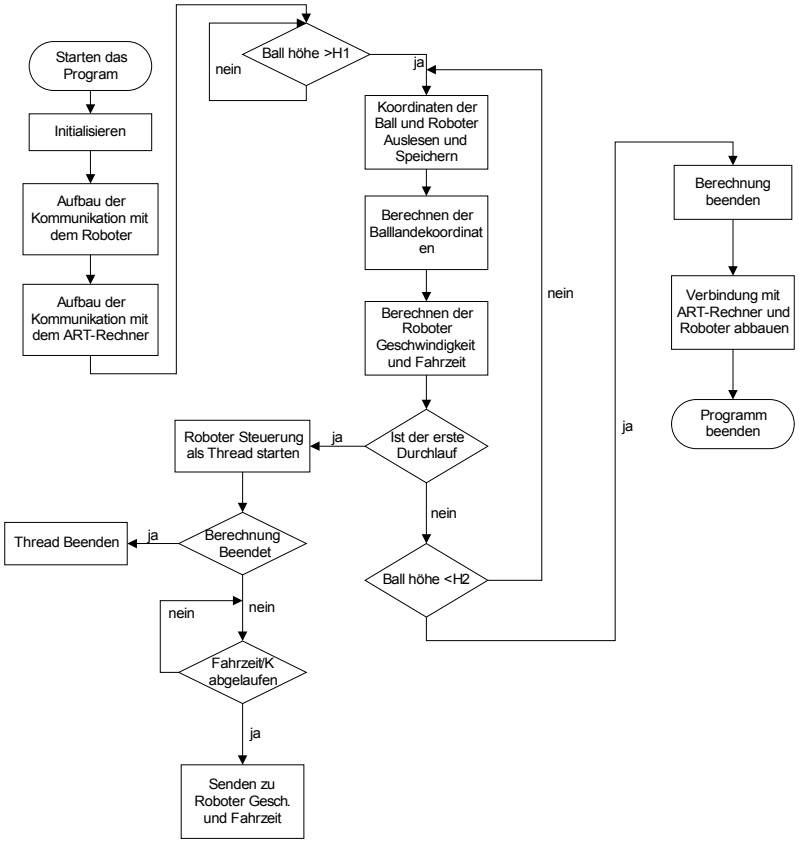


Abbildung 4.6: Programmablauf der ganze Programm

Für die Kompensation von dem Fehler beim Schlupf der Roboter-Räder müßte der Fangkorb (siehe Abbildung 2.7) so etwa 3 mal großer als der Ball selbst sein. Der Fangkorb muss stabil auf dem Roboter sitzen, denn auf dem Fangkorb ist der Roboter-ART-Body befestigt. Wenn der Fangkorb beim Fahren des Roboters stark wackelt, werden bei Berechnung der Roboter-Geschwindigkeit und -Fahrzeit größere Abweichungen ausgewiesen. Außerdem muss der Roboter-ART-Body so auf dem Fangkorb befestigt sein, dass auch die unteren Kameras des ART-Systems zu jeder Zeit den Body identifizieren können. Deshalb ist der Roboter-ART-Body etwa 0.1 m in der Mitte über dem Fangkorb befestigt. Dass der Roboter-ART-Body in der Mitte des Fangkorbes befestigt ist, hat den Nachteil dass der Ball auf diesen fallen kann. Der Versuch den Body auf der hinteren oder auf der vorderen Seite des Fangkorbes zu plazierten, in dem die Differenzen gebildet werden, ist auf Grund des ständigen Schlupfs der Roboterräder auf dem Fußboden gescheitert. Bei einem anderen Bodenbelag mit größerer Bodenhaftung kann diese Methode möglicherweise funktionieren.

Als erstes muß die Kommunikation zwischen ART-Rechner und Steuer-Rechner hergestellt werden (siehe Referenz (fastlabinc)).

Zm Herstellen der Kommunikation mit dem Tracking-Rechner,dem Auslesen der Ball- und Roboter-Koordinaten so wie dem Ausführen des ganzen Programmablaufs wird die Klasse $CTrack$ benutzt. Diese Klasse beruht zum Teil auf der eine Klasse, die der Assistent der MMLab Andreas Ißleib als Orientierung zur Verfügung gestellt hat. Und ebenso auf der von meinem Betreuer Professor Renz zur Verfügung gestellten ART-Struktur der UDP-Pakete. Die Klasse $CTrack$ beinhaltet zwei Funktionen, die Funktion $UDPClient()$ die erst die Kommunikation mit dem ART-Rechner herstellt und in einer Endlosschleife die zweite Funktion $parseFrame()$ aufruft. In der Funktion $parseFrame()$ wird erst die Berechnung gestartet, wenn der Ball eine bestimmte Wurfhöhe überschreitet. Dann folgt das Auslesen der Koordinaten der einzelnen Bodies und das Speichern dieser Koordinaten in der Klasse $CUDPDaten$. Danach, wenn die Berechnung schon gestartet ist, werden die einzelnen Funktionen der entsprechenden Klassen zur Berechnung der Parabelregression und die Nullstellen der quadratische Polynom gestartet. Es folgt die Berechnung der Gerade-Regression und danach der voraussichtlichen Ballslandekoordinaten berechnen. Der Start der Robotersteuerfunktion, diese Funktion wird permanent mit die aktuellen Roboter-Geschwindigkeit und Roboter-Fahrzeit versorgt. Diese Berechnungen werden so lange fortgesetzt bis die Berechnung beendet ist, und so mit auch die Beendigung das ganzes Programm. Diese Bedingung tritt ein wenn der fliegende Ball eine bestimmte Wurfhöhe unterschreitet.

In dieser Aufgabe bezieht sich die Ausrechnung der Ballslandekoordinaten nicht nur auf die aktuellen Ballkoordinaten sonder auch die vorherigen. So wie in Unterkapitel 4.1 beschrieben wird, handelt es sich hier um ein dynamisches System. So ein System braucht Platz wo Daten sicher gespeichert werden und bei Bedarf in Echtzeit aus- und einlesen kann.

Aus diesem Grund ist hier eine Klasse $CUDPDaten$ vorgesehen, die die ART-Daten von den UDP-Paketen ausliest und in Arrays speichert. Diese Klasse muss die Möglichkeit bieten dass diese Daten auch von einer anderen Stelle ausgelesen werden können, und das in Echtzeit. Die Funktionen die diese Klasse unbedingt haben muss, sind:- Das Speichern in dynamisches Array den UDP-Koordinaten des Roboters und des Balls, die Funktionen zum Auslesen dieses dynamischen Arrays an bestimmten Stelle in dem Array, nicht aber über die Grenzen diese Array hinaus, die Funktionen die die Länge dieses Arrays verändern, vergrößern und die Konstruktoren so-wie Destruktoren.

Die Klasse die die Nullstellen des quadratischen Polynom berechnet, ist als nächstes entworfen. Diese Klasse muss eine 3x3 Matrix beinhalten, wie in der Formel 3.22 gezeigt ist. Diese Matrix muss auch entsprechend so implementiert werden. Auch zwei Arrays mit Länge 3 auf der rechten Seite des Gleichungssystems 3.22 und eines für das Ergebnis der Koeffizienten des quadratischen Polynoms. Die Klasse muss folgende Funktionen beinhalten:- zur Initialisierung der Matrix wie in der Formel 3.22 und zur Initialisierung der Vektoren ebenso wie in der Formel 3.22:- eine Funktion zur Berechnung der Untermatrix, die bei Berechnung der inversen Matrix benötigt wird:- eine Funktion zur Berechnung der Determinante,:- eine

Funktion zur Berechnung der inversen Matrix und schließlich der Konstruktor und der Destruktor. Diese Klasse hat noch mehrere Funktionen die nur für den Testzweck sind, und die von der (*.ini) Datei mit Zweck von Benutzter eine bestimmte Initialisation der Variablen möglich machen.

Die nächste Klasse approximiert die Gerade die die Parabel auf der X-Z Projektionsebene beschreibt, und berechnet die Robotergeschwindigkeit auf X- und Z-Richtung so wie die Fahrzeit der Roboter. Diese Klasse muss die folgende Funktionen haben:- Initialisierung der Matrix wie in der Formel 3.25 und eine weitere Funktion zu Initialisierung des Vektors auch von der selben Formel 3.25:- ein Funktion zur Berechnung der Gerade-Koeffizienten:- die Funktion, die auf Basis der Ballslandekoordinaten und der aktuellen Roboterkoordinaten die Robotergeschwindigkeit auf X- und Z- Richtung so wie die Roboterfahrzeit berechnet. Diese Klasse muss auch Funktionen haben, die die berechnete Geschwindigkeiten und Fahrzeit der Roboter weitergeben können. Diese Klasse hat auch andere Funktionen, die zur Initialisierung benötigt werden.

Die Klasse, die der Roboter steuert $CRobotCom$ muss erst von der Klasse Com aus der Robotino API und gleichzeitig von der Klasse $CThreadInter$ die ein Interface ist abgeleitet werden. Die Klasse $CThreadInter$ ist von der Klasse $CMThread$-(diese Klasse ist ein Thread mit ein abstrakte Funktion) abgeleitet. Diese doppelte Vererbung ist nötig, weil die Robotino-API eingebunden werden und gleichzeitig eine bestimmte Funktion von diese Klasse als Thread gestartet werden muss, die den Roboter steuert und seine aktuelle Geschwindigkeit und Fahrzeit einliest. Diese Klasse besitzt folgende Funktionen $init()$, zur Initialisierung der el. Motoren und zum Aufbau der Ethernet-Verbindung mit dem Roboter Server:- die Funktion $connectedEvent()$ signalisiert, wenn die Verbindung mit dem Roboter zu Strande gekommen ist:- die Funktion $connectionClosedEvent()$ signalisiert, wenn die Verbindung mit dem Roboter unterbrochen ist:- die Funktion $drive()$ steuert zusammen mit der Funktion $execute()$ den Roboter:- die Funktion $execute()$ ist die Implementierung der abstrakte Funktion der Klasse $CMThread$, die Funktion $destroy()$ unterbricht die Verbindung mit dem Roboter.

Die Klasse $CMThread$ ist eine abstrakte Klasse. Von dieser Klasse können anderen Klassen abgeleitet werden. Diese abgeleiteten Klassen müssen die Funktion $execute()$ implementieren, was als Abstrakt in dieser Klasse deklariert ist und durch die Funktion $start()$ die implementierte Funktion als Thread nebenläufig startet. Die Kern Funktion in dieser Klasse ist der Windows-System Funktion $beginthreadex()$. Mit Hilfe dieser Klasse kann der Roboter im Hintergrund als Thread laufen. Diese Klasse hat die folgende Funktionen: $start()$ zum starten den Thread. In der Funktion $start()$ wird die Windows-System Funktion $beginthreadex()$ aufgerufen, was einen Thread startet. Es wird die als Startadresse die Adresse der Funktion $entryPoint()$ eingegeben. In der Funktion $start()$ wird auch des Thread Priorität gegeben, hier als Normal-Priorität. Die Funktion entryPoint wird in der Windows-System $beginthreadex()$ aufgerufen und die ruft die Funktion $run()$, die wiederum der Funktion $execute()$ aufruft. Die Funktion $run()$ ruft die Abstrakte Funktion

$execute()$ auf, die in die abgeleitete Klasse nach eigenem Wunsch implementiert werden muss.

Die Klasse „CThreadInter“ ist eine Klasse die als Interface dienen muss, weil eine direkte Ableitung von der Klasse „Com“ und „CMThread“ nicht möglich war, da diese eine abstrakte Klasse war. Deshalb war es nötig die abstrakte Funktion in diese Klasse erst (leer) zu implementieren. Im nächsten Unterkapitel werden die Flußdiagramme und die Funktionalität der Flußdiagramme behandelt.

Die Klasse „IniReader“ ist dafür da, das Programm, bevor es gestartet wird, zu Konfigurieren. Wann die Berechnung gestartet wird, die Wurfhöhe zum Starten und Beenden den Berechnung festgelegt. Die maximale Robotergeschwindigkeit und die Roboter Zeitkonstante die dafür benutzt wird die Roboter Zeit zu unterteilen. Die maximale zugelassene Fahrzeit für den Roboter, diese ist immer ein bisschen größer als die wirklich maximal berechnete, damit der Roboter früher in Bewegung gesetzt werden kann. Die Variablen initialisieren die zum Transformieren der Nullstelle der Koordinatensystem benötigt werden. Die Kommunikationvariablen der IP-Adressen des ART-Rechners und des Roboter-Servers, der Port Nummer und die Internet Protokollfamilie UDP/IP oder TCP/IP die benutzt wird, werden festgelegt.

Und schließlich gibt es ein C++ File, in dem die $main()$ Funktion implementiert ist. In der main-Funktion werden die Objekte und deren Referenzen erzeugt die Kommunikation mit dem Roboter hergestellt, die Funktion $UDPClient()$ der Klasse $CTrack$ aufgerufen und so das Programm gestartet. Nachdem die Berechnung beendet ist, wird die Verbindung mit dem Roboter abgebaut, und die Testfunktionen der einzelnen Klassen aufgerufen.

Der ganze Ablauf des Programms geschieht in der Funktion parseFrame() der Klasse $CTrack$. In dieser Klasse wird die Berechnung gestartet, nachdem der Ball eine bestimmte Wurfhöhe überschritten hat. Diese Wurfhöhe wird in der Klasse $IniReader$ bevor die Berechnung im Programm anfängt festgelegt. Mittels eines Flags wird die Überschreitung diese Wurfhöhe vom Ball gespeichert. In eine Schleife werden dann die alle Bodies der UDP-Pakete den ART-Rechner ausgelesen und in dynamischen Arrays der Klasse $CUDPDaten$ gespeichert. Danach wird überprüft ob der Ball auch eine bestimmte Wurfhöhe nicht unterschritten hat ist es der Fall wird die Berechnung beendet und das ganzes Programm auch. Ist die Berechnung noch nicht beendet dann werden die einzelne Funktionen der Klassen aufgerufen. Als erstes wird die Funktion der Matrix wie in Formel-3.22 aufgebaut und danach auch der Vektor, der auf der rechte Seite diese Formel steht. Die Determinante der Matrix wird nach „Laplace“ entwickelt, danach wird die Inverse Matrix mit Hilfe der Determinante berechnet. Danach wird auch das Gleichungssystem gelöst und das Ergebnis sind die Koeffizienten des quadratischen Polynoms. Dieses quadratischen Polynoms werden dann die Nullstellen berechnet, die auf der Höhe des Roboter-Fangkorb-Niveaus liegen. Die Koeffizienten der projektierte Gerade der Parabel auf der X-Z Projektionsebene (Fußboden) werden auf der selben Weise wie der Parabel-Koeffizienten berechnet. Nur hier ist der Matrix 2x2 und nicht 3x3 wie bei dem quadratischen Polynom. Die Nullstelle der Parabel wird

in der Geradengleichung gesetzt und diese Gleichung nach X oder Z Koordinate aufgelöst. Nachdem die Ballslandekoordinaten berechnet sind, wird die aktuelle Position des Roboters ausgelesen und von den beiden Koordinatenpunkten (Ball- und Roboter-Position) nach Pythagoras die Entfernung und danach auch der Winkel zwischen der Roboter und der Ball berechnet (siehe Abbildung3.2.3). Mit diesem Winkel wird die Geschwindigkeit in X- und Z-Richtung des Roboter berechnet. Die Roboter-Fahrzeit wird von über Entfernung und die maximale Robotergeschwindigkeit berechnet. Diese Berechnungen werden ununterbrochen durchgeführt, wenn dies zum ersten Mal passiert, wird die Robotersteuerung-Funktion als Thread(nebenläufig) gestartet. Der Thread wird immer für sehr kurze Zeitintervalle unterbrochen, dann werden dem Roboter die neu berechnete Geschwindigkeit und Fahrzeit an der Robotersteuerung-Funktion übergeben. Die Berechnung wir so lange durchgeführt, bis der Ball eine bestimmte Wurfhöhe in seiner Flugbahn unterschreitet, dann wird das Programm beendet.

4.2.1 Flussdiagrammen

Hier werden die Flußdiagramme Programmablauf-Funktionsablauf dargestellt. Beim Starten des Programms werden erst die benötigten Objekte und die zugehörigen Pointers erzeugt. Nach Erstellung der Verbindung mit dem Robotino-Server und dem Initialisieren der Elektromotoren der Roboter-Robotino wird die Funktion $UDPClient()$ der Klasse $CTrack$ aufgerufen. Mit dem Aufruf dieser Funktion wird das Programm gestartet. Nachdem das Programm beendet ist, läuft die $main()$ Funktion weiter. Es folgt das dazugehörige Flußdiagramm.

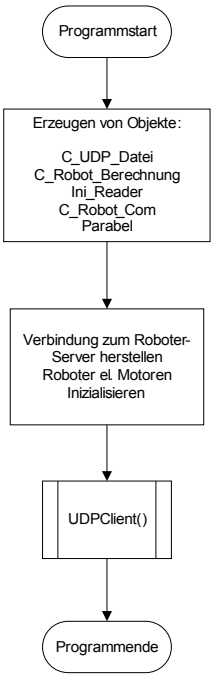


Abbildung 4.7: main Flußdiagramm

In der Klasse $CTrack$ wird der weitere Programmablauf stattfinden. Diese Klasse enthält zwei Funktionen, $UDPClient()$ und $parseFrame()$. In der Funktion $UDPClient()$ wird erst die Verbindung zum Tracking-Rechner des ART-Systems hergestellt. Ein Puffer zur Speicherung der von Tracking-Rechner gesendete UDP-Pakete, wird erzeugt. Es wird ein Socket erzeugt. Die Funktion $UDPClient()$ läuft weiter in einer Endlosschleife. In diese Schleife wird auch die Funktion $parseFrame()$ aufgerufen. In eine „if“ Bedingung wird geprüft, ob diese Endlosschleife beendet wird und damit das ganzes Programm auch beenden. Es folgt das dazugehörige Flußdiagramm.

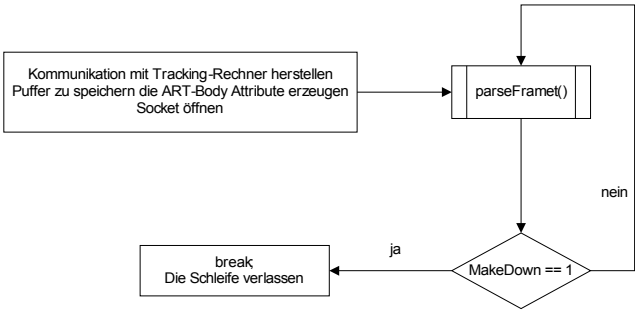


Abbildung 4.8: UDPClient Flußdiagramm

In der Funktion „parseFrame()“ (siehe Abbildung4.9) werden die Koordinaten aus dem „buffer“ gelesen und auf das Maß Meter Normiert. Der Koordinatenursprung wird verlegt. Wenn der Ball eine bestimmte Wurfhöhe überschreitet, wird mit der Berechnung begonnen und eine Startmarke ist gesetzt. Es werden die Body-Id-s gelesen. Wenn die Ball-Id gelesen wird, wird erst geprüft ob die Koordinatendaten brauchbar sind. Wenn diese Koordinatendaten gültig sind werden sie gespeichert. Wenn die Koordinatendaten nicht gültig sind werden nur Nullen geschrieben. Diese Maßnahme wird getroffen, da das ART-System manchmal den Roboter-Body identifizieren kann, nicht aber den Ball-Body, oder umgekehrt. So bleiben alle Datensätze auf der selben Länge. Wenn der Ball eine bestimmte Wurfhöhe unterschreitet z.B. 0.5 Meter (0.5 ist die Höhe des Fangkorbs des Roboters) wird das Programm beendet und eine Stopmarke wird gesetzt. Wenn die Roboter-Id gelesen wird, wird erst geprüft, ob die Koordinatendaten brauchbar sind. Wenn diese Koordinatendaten die richtigen sind, werden sie gespeichert. Wenn die Koordinatendaten unbrauchbar sind werden nur Nullen geschrieben. Ist die Startmarke gesetzt und die Stopmarke noch nicht , werden die Unterprogramme aufgerufen. Erst wird das lineare Gleichungssystem zur Berechnung der Parabel-Koeffizienten gelöst. Die Nullstellen der Parabelgleichung werden berechnet und die richtige der beiden Nullstellen ausgewählt. Daraufhin wird die Projektionsgerade der Parabel auf der die selbe Projektionsebene berechnet.Wenn der Roboter-Thread kurzzeitig unterbrochen wird, werden die voraussichtlichen Ballslandekoordinaten, die Roboter-Geschwindigkeit auf der X- und Z-Achse sowie die Roboter-Fahrzeit berechnet. Sind die neu berechneten Geschwindigkeiten auf X- und Z-Achsen sowie die Fahrzeit annehmbar, werden sie an den Roboter weitergegeben. Sonst fährt der Roboter mit der alten X- und Z- Achse Geschwindigkeit und alten Fahrzeit. Zuletzt wird bei Übergabe der neuen Robotergeschwindigkeit und Roboterfahrzeit überprüft, ob die Robotergeschwindigkeit oder Roboterfahrzeit gültige Werte haben. Ist es nicht der Fall wird die Projektionsebene gewechselt.

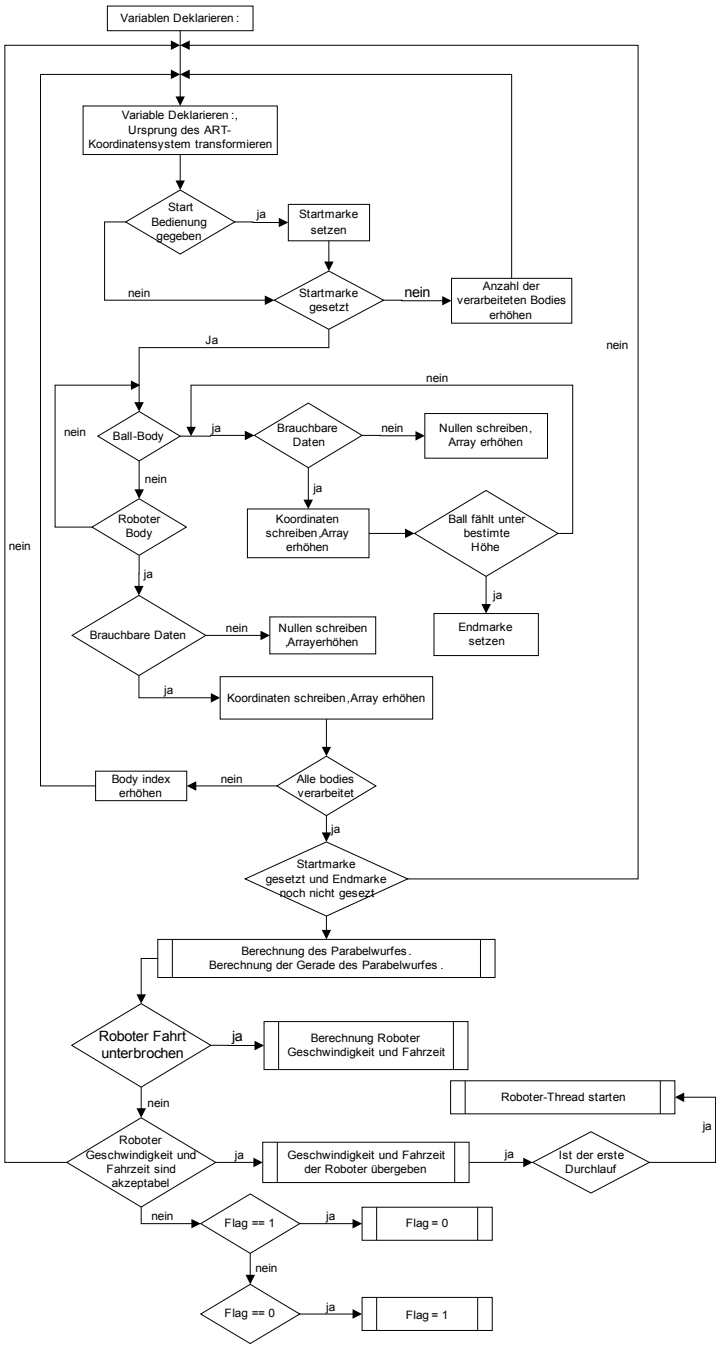


Abbildung 4.9: parseFrame Flußdiagramm

4.2.2 Auswahl der Programmiersprache und IDE

Die Auswahl der IDE in dieser Diplomarbeit ist eng mit der Roboter-Robotino-API verbunden. In den Roboter ist als Steuerrechner ein PC-104 eingebaut. Auf diesem ist dem Realtime Debian-Linux installiert. Wenn für den Roboter Programmen entworfen werden, ist es nicht notwendig auf seinem Embedded PC das Programm zu schreiben. Das ist auch eine Möglichkeit den Roboter zu programmieren ist aber nicht sehr komfortabel. Für diesen Zweck hat die Firma Festo API zum Entwurf von Programmen zur Verfügung gestellt. Die API ist für das Betriebssystem-Windows und die IDE- MSVS- 2003,2005,2008 so wie für die Programmiersprachen C/C++ entworfen. Für das Betriebssystem Linux gibt es auch eine API, diese ist für den GCC-4.1 Compiler entworfen. Dabei müssen die Makefiles aber an den Programmentwurf angepasst werden.

Die Auswahl ist zwischen C und C++ beschänkt. Wenn die Programmiersprache C als Entwicklungssprache ausgewählt wird muss mit viele globalen Variablen in den Header-Files gerechnet werden, was die Übersicht in dem ganzen Programm einschränkt. Durch eine Kapselung der Daten verbessert sich die Übersichtlichkeit des Programms. So die Wahl fiel an der Programmiersprache C++. Als IDE ist Microsoft Visual Studio-2005 ausgewählt.

4.3 Implementation

4.3.1 Anforderungen bei Implementation

Der Tracking-Rechner sendet die UDP-Pakete mit eine Geschwindigkeit von $60\,\mathrm{Hz}$ plus der Verzögerung(timedelay) des ART-Systems. Die Daten(Koordinaten) müssen schnell eingelesen werden damit keine Werte verloren gehen können. Die Rechenzeit zur Verarbeitung den Daten muss so klein wie möglich sein. Eine übermäßige dynamische Speicherallokation muss aus diesen Gründen vermieden werden, denn der Rechenaufwand dafür steigt mit der Anzahl der Messwerte.

4.3.2 Implementierung

In der Implementation der entworfenen Klassen ist Anforderung auf Schnelligkeit und Kapselung der Objekte als Premiere festgelegt. So wenig wie möglich Speicherallokation und keine globale variablen. Alle programmiere Strukturen wie z.B. Matrizen wurden schon am Anfang der Programm auf der Heap gelegt. Sie werden nur neu überschrieben. Die Überschreibung dieser Strukturen auf dem Heap kostet auch Zeit, aber viel weniger als für diese Stuckturen neu Speicherplatz zu reservieren und dann auch freizugeben. Diese Strukturen werden am Ende des Programms auf dem Heap gelöscht. Ausnahme sind die dynamischen

Arrays für Ball- und Roboter- Koordinaten, und dynamische Arrays, die für Testzwecken benutzt wurden.

Bei der Implementierung hätten auch bekannte C++ Bibliotheken, wie „STL“ oder „Boost“, benutzt werden können. Diese stellen Datenstrukturen, z.B. Matrizen und Vektoren, zur Verfügung. Diese Bibliotheken wurden allerdings in dieser Arbeit nicht benutzt, für „Boost“ wäre eine Einarbeitungszeit nötig gewesen. Alle in dieser Arbeit benötigten Datenstrukturen sind nativ in der Programmiersprache C++ implementiert.

Um die besseren Laufzeit zu erzielen wurden in den Programmen $CallbyReferenz$ Techniken angewendet, so müssten die Objekte nicht kopiert werden, (sieh (Peter Prinz, 2002)).

In die Abbildung 4.6 ist der Ablauf des ganzes Programms zu sehen. Die Objekte und die Pointer hierzu werden in $main()$ Funktion erzeugt. In der $main()$ Funktion werden auch die Variablen in den Klassen durch die Klasse $IniReader$ initialisiert. Der Kommunikationsaufbau mit dem Roboter-Server wird durch die Funktionen der Robotino-API aufgebaut. Die statische Funktion $UDPClient$ wird in der $main()$ Funktion aufgerufen und damit die ganze Kontrolle des weiteren Programmablaufs an diese Funktion übergeben. In dieser Funktion wird die Verbindung zum ART-Rechner aufgebaut. Die Funktionen zum Aufbau dieser Verbindung sind aus der Klasse $Socket$ die auch die Vaterklasse in die C++ Datei $PracticalSocket$ ist, vererbt. Diese Kommunikation wird durch die Funktion $resolveService()$ ausgeführt, die als Parameter die Serverport und die Art vom IP Protokoll erwartet. Durch eine Instanz der Klasse $UDPSocket$ wird ein neuen Socket erzeugt. Durch die Funktion $recvFrom()$ der Klasse $UDPSocket$ werden die Ball- und Roboter-Koordinaten durch die UDP/IP Paketen vom ART-Rechner empfangen. Mit der statische Funktion der Klasse $CTrack$ wird die Auslesen die Koordinaten von den UDP/IP Paketen die aktuelle Berechnung den Ballslandekoordinaten und so die Robotersteuerung durchgeführt.

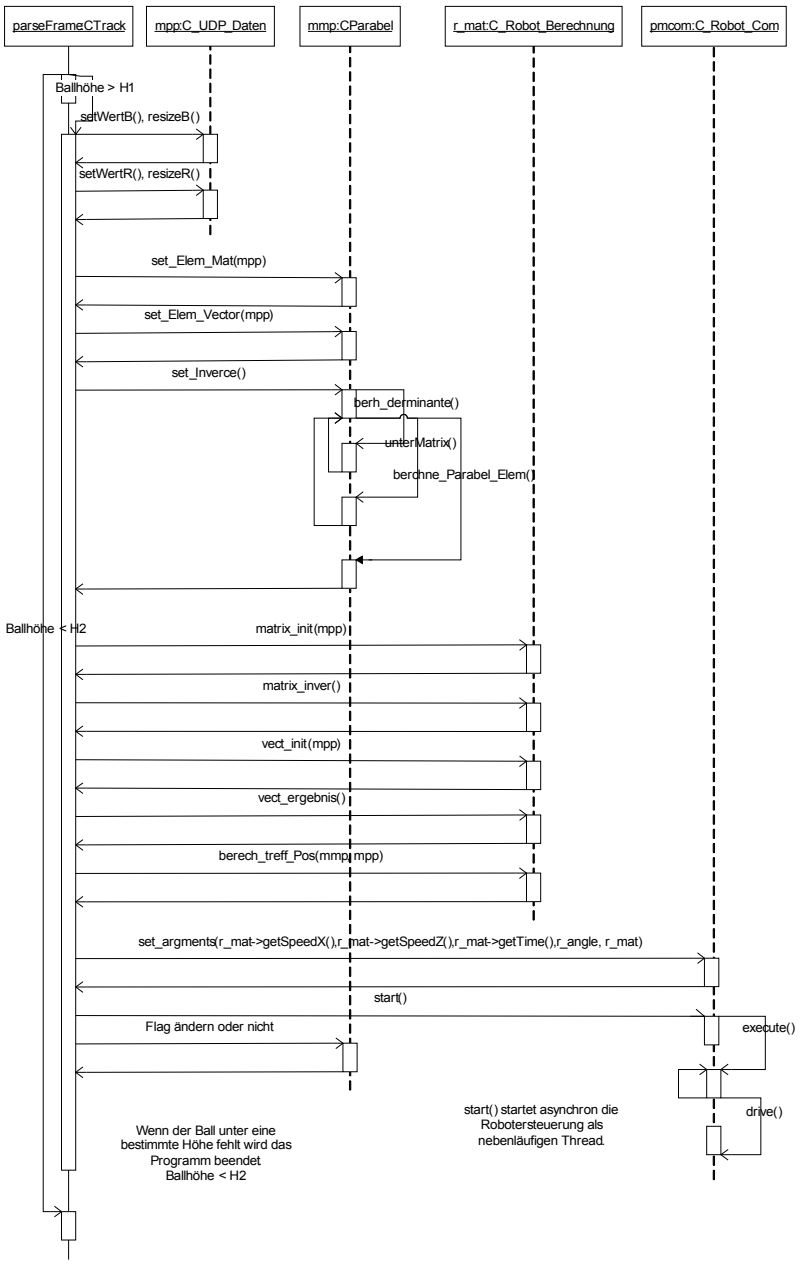


Abbildung 4.10: Sequenzdiagramm des Programmablaufs in ParceFrame

In der statischen Funktion $ParseFrame()$ wird der sequentielle Aufruf der Funktionen erfolgen. Durch diesen Funktionen werden die nötigen Parameter berechnet damit das Programm erfolgreich terminiert. In Abbildung 4.10 ist dieser Aufruf gezeigt. Wenn der Ball eine bestimmte Wurfhöhe überschreitet, wird die Berechnung gestartet. Die Pointer der in der main-Funktion erzeugten Objekte werden als Parameter in die Funktion $ParceFrame()$

übergeben, sie werden dazu benutzt um bei jeweiligen Objekten die entsprechenden Funktionenaufrufe zu ermöglichen.

Die Koordinaten des Balls und des Roboters werden eingelesen und gespeichert. Danach werden die Funktionen zu Berechnung der polynom Regression aufgerufen. Die quadratische Regression wird nach der Formel (siehe 3.22) implementiert. Dazu werden eine Matrix und ein Array gebraucht. Die Matrix für die quadratische Regression wird initialisiert. Der Vektor für die quadratische Regression wird ebenfalls initialisiert. Die inverse Matrix wird berechnet. In der Funktion $setInverce()$ wird dann die Funktion $berchneParabelElem()$ aufgerufen, diese Funktion berechnet dann die voraussichtlichen Nullstellen des quadratischen Polynoms und wählt aus, welche von den beiden der richtige ist. Jetzt ist die richtige Nullstelle bekannt.

Die lineare Regression der Gerade wird nach der Formel (siehe 3.25) implementiert. So wie in der quadratischen Regression werden auch hier eine Matrix und ein Vektor gebraucht. Als Nächstes wird die Regression der Gerade(Projektion der Parabel auf dem Boden), anschließend die Z-Koordinate des Balls berechnet. Die Matrix für die gerade Regression wird durch Aufruf der Funktion matrixini initialisiert. Der invertieren diese Matrix wird mit der Funktion matrixinver durchgeführt. Durch den Aufruf der Funktion vectinit wird der Vektor Initialisiert. Berechnen den Koeffizienten der Gerade wird durch die Funktion $vectergebnis()$ durchgeführt.

In der nächsten Funktion $berechTreffPos()$ werden die Z- oder X- Komponente der Ballslandekoordinaten berechnet, dann auch die Robotergeschwindigkeit auf X- und Z-Richtung und Roboter-Fahrzeit berechnet. Das Berechnete Robotergeschwindigkeit und Roboterfahrzeit werden an der Robotersteuerung-Funktion $drive()$ der Klasse $CRobotCom$ durch die Funktion $setAargments()$ von der selbe Klasse übergeben. Damit ist der Roboter mit die aktuelle Geschwindigkeit und Fahrzeit versorgt. So dass wenn er vom Kurs abweicht mit der nächste Parameter(X-Z Geschwindigkeit und Fahrtzeit) Übergebe kann wieder in der richtige Richtung gelenkt werden.

Das Roboter Steuerungs-Programm wird als nebenläufig(Thread) gestartet und läuft im Hintergrund bis zum Ende das Programm weiter. Die Funktion $start()$ wird aufgerufen damit der Roboter-Steuerung als Thread gestartet wird, dieser Thread muss nur einmal gestartet werden. Nachdem Funktion $start()$ aufgerufen wurde, wird durch diese die Funktion execute gestartet und sie ruft die Funktion $drive()$ auf. In Funktion $drive()$ wird die Funktion $setVelocity()$, so wird der Roboter bekannt gemacht wie er fahren soll.

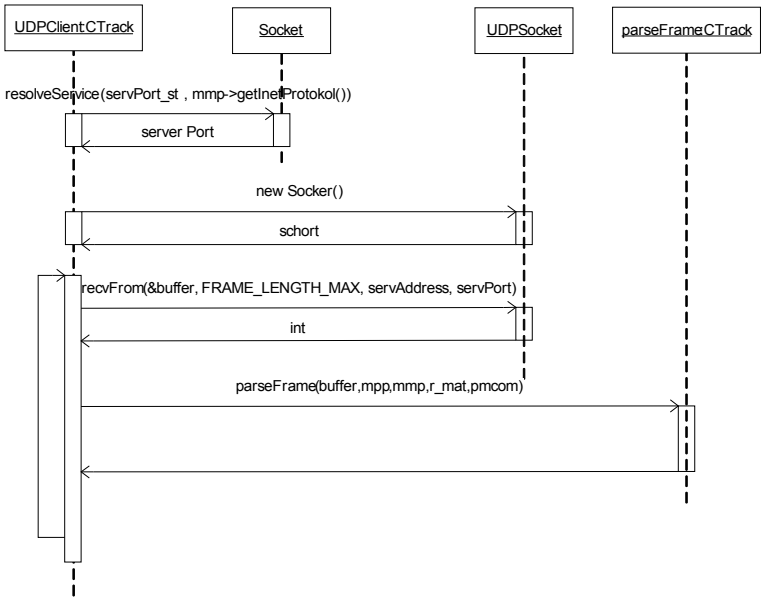


Abbildung 4.11: Sequenzdiagramm der Programmablauf in UDPClient

In der statischen Funktion UDPClient der Klasse CTrack wird die Verbindung zum ART-Rechner aufgebaut. Diese Kommunikation-Socket-Funktionen sind von vorigen Projekten, die mit ART-System gearbeitet haben, übernommen. Der Serverport wird erhalten mit der Funktion $resolveService()$. Mit dieser Funktion wird auch angegeben mit welcher IP-Familie die Kommunikation erfolgen wird. Diese Klasse ist aus der Vaterklasse Socket der C++ Datei $PracticalSocket$. Mit einem Pointer der Klasse wird einen neuen Socket erzeugt. Ein Pointer auf den Anfang den UDP/IP-Paketen wird mit der Funktion $recvFrom()$ der Klasse UDPSocket erfolgen. Diese Pointer auf den UDP/IP-Paketen wird als Parameter an der statische Funktion parseFrame der Klasse CTrack zum Einlesen der Ball- und Roboter-Koordinaten übergeben. Es folgt der Aufruf der Funktion „recvFrom“ und nach dieser auch der Funktion $parseFrame$ erfolgt in eine Endlosschleife. Wenn der Ball unter eine bestimmte Wurfhöhe fählt dann wird eine Marke $MarkeDown$ gesetzt, dann wird die Berechnung und auch somit auch das Programm beendet.

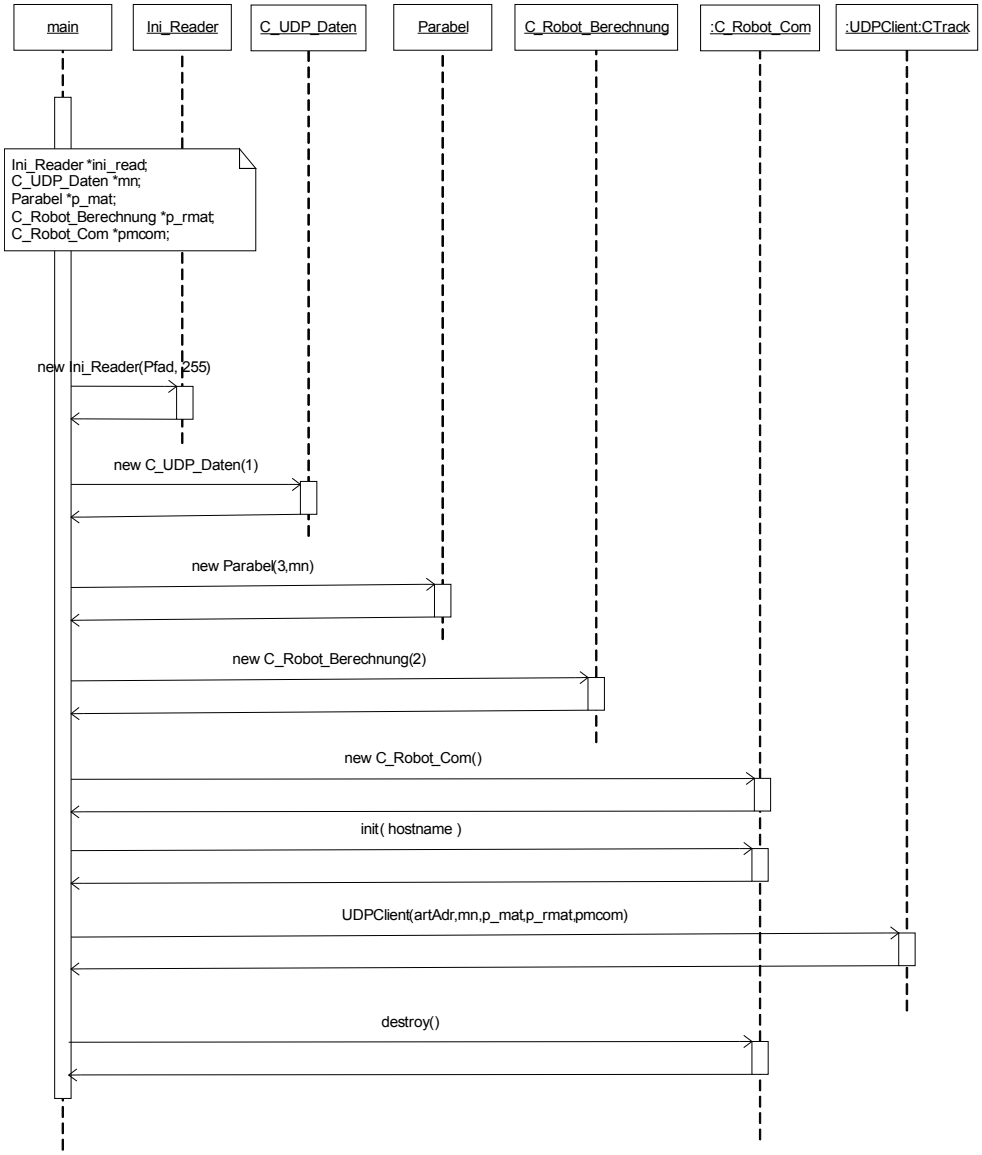


Abbildung 4.12: Sequenzdiagramm des Programmablaufs in main Funktion

In der main-Funktion wird erst die aktuelle Directory gelesen und an der Konstruktor der Klasse $IniReader$ übergeben. Danach werden die Pointers alle Klassen deklarieren. Diese werden benutzt damit nicht die Objekte kopiert werden sondern nur ihren Adressen. Ein Pointer ist nur vier Byte groß, damit wird ein besseres Zeit-Performance erzielt. Die folgende Pointer auf Objekten werden deklarieren.

- Pointer der Klasse IniReader.

- Pointer der Klasse CUDPDaten.
- Pointer der Klasse Parabel.
- Pointer der Klasse CRobotBerechnung.
- Pointer der Klasse CRobotCom.

Die Verbindung zum Robotino-Server wird aufgebaut durch den Aufruf der Robotino-API-Funktion init. In der $main()$ wird eine Informationausgabe erfolgen, die zeigt dass das Programm läuft und die Verbindung erfolgreich durchgeführt ist. Die statische Funktion UDPClient der Klasse CTrack wird aufgerufen und als übergabe Parametern bekommt diese Funktion die Adresse des ART-Rechners(damit die Verbindung zu der ART-Rechner zustande kommt) und auch die Pointer zu den Instanzen der Klassen CUDPDaten, Parabel, CRobotBerechnung, CRobotCom. Die Klasse IniReader ist nur zur Initialisieren und spielt keine Rolle bei der Berechnung deshalb wird ihre Adresse nicht als Parameter übergeben. Nachdem die Berechnung beendet ist, wird main-Funktion wieder Rechenzeit bekommen. Danach wird die Verbindung zu dem Robotino-Server abgebaut.

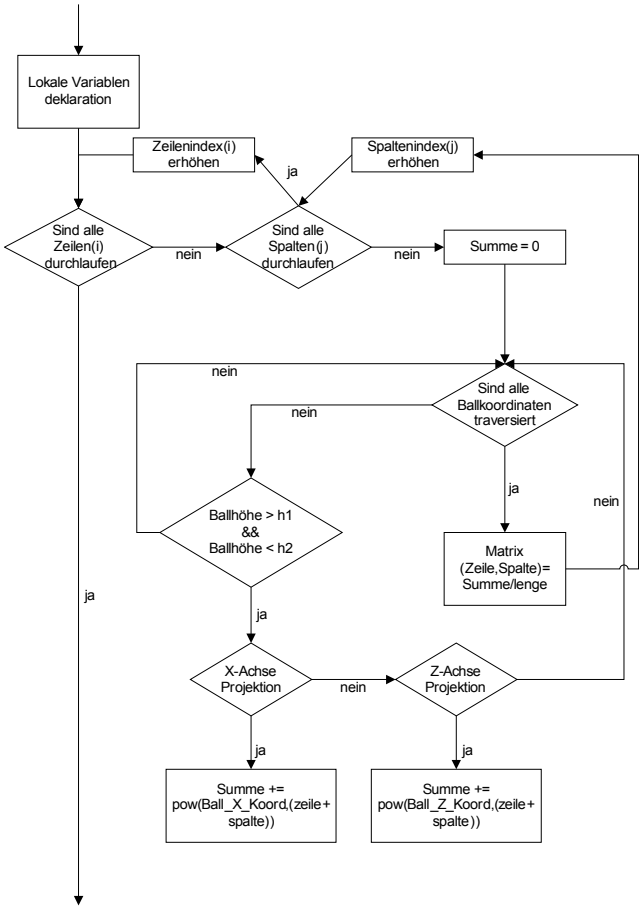


Abbildung 4.13: Implementation der Regressionmatrix

Mit der Funktion $setElemMat()$ (siehe Abbildung 4.13) der Klasse Parabel wird die Matrix wie in der Formel (siehe 3.22) im Kapitel-technische Analyse mit Werte initialisiert. Erst wird in zwei for Schleifen die Matrix traversiert, diese Matrix ist im Konstruktor erzeugt und befindet sich auf dem Heap. Diese Matrix wird nur jedes mal überschrieben, aber es wird keine Speicherallokation durchgeführt. Mit eine dritte for Schleife werden die Koordinaten-vektoren der X- und Z-Koordinaten-bahn des Balls traversiert.Eine Überprüfung wird durchgeführt, ob die Werte dieser Koordinaten gültig sind. Bevor die dritte for Schleife gestartet wird, wird der Akkumulator (die Variable summe) mit NULL initialisiert. Mittels ein Flag wird geprüft auf welche Projektionsebene die Regression durchgeführt wird. Wenn der Flag gleich 0 ist wird die X-Y Projektionsebene ausgewählt. Dann werden die Potenzen alle X-Koordinaten-Werte der Ballbahn berechnet. Die Potenzen sind von der aktuellen Zeile und Spalte der Matrix abhängig und können Werte zwischen 0 und 4 bekommen. Nachdem die entsprechende Koordinaten der Vektor bis zum Ende traversier sind wird der Akkumulator(summe) durch die Länge des traversierten Koordinaten-vektors dividiert, so wird ein Mittelwert errechnet. Mit dieser Mittelwert wird dann der entsprechende

Matrix-Index initialisiert.

Diese Matrix wurde auch mit Matlab gebildet und mit der, die in C++ implementierten Matrix verglichen, die beide Matrizen sind identisch. Die Ergebnisse sind im Kapitel Test und Bewertung zu finden.

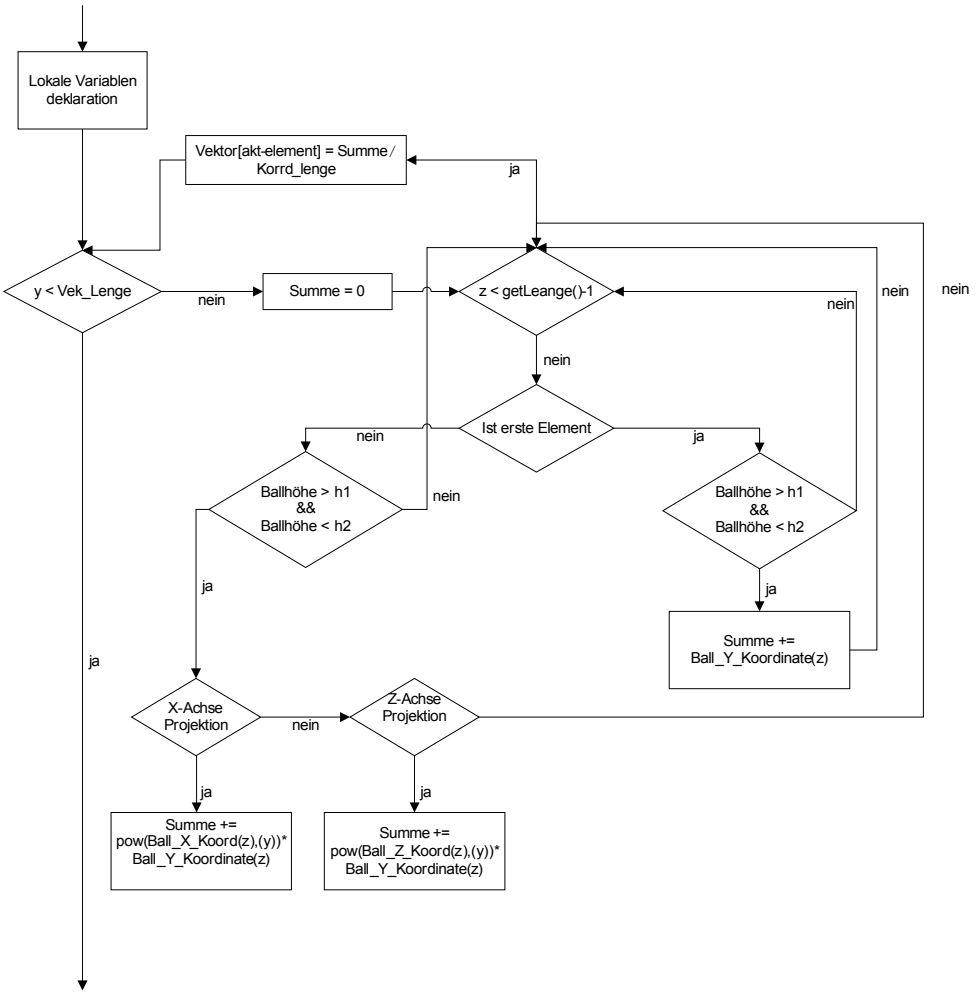


Abbildung 4.14: Implementation der Regressionsvektor

Mit der Funktion $setElemVector()$ (siehe Abbildung 4.14) der Klasse Parabel wird der Vektor von der Formel (siehe 3.22) im Kapitel-technische Analyse mit Werte initialisiert. Erst wird mit einer for Schleife der Vektor traversiert. Der Akkumulator(summe) wird mit NULL initialisiert. Mit einer zweiten Schleife wird die entsprechenden Ballbahn Koordinaten Vektor traversiert. Wenn das erste Element des Vektors initialisiert wird, wird der Y-Ball Koordinaten Vektor aufsummiert. Vorher wird geprüft, ob die Werte der laufende

Y-Ball Koordinate gültige Werte sind. Wird nicht der erste Element der Vektor initialisiert dann wird erst überprüft auf welcher Projektionsebene die Regression durchgeführt wird. Ist dies die X-Y Projektionsebene, werden die Potenzen aller X-Koordinaten-Werte der Ballbahn berechnet. Ist auf Z-Y Projektionsebene, werden die Potenzen aller Z-Koordinaten-Werte der Ballbahn berechnet. Diese Berechneten Potenzen-werte werden dann mit dem Y-Koordinaten Wert mit dem selben Index multipliziert und dann durch die Länge der Koordinaten Vektor dividiert, so ernsten auch ein Mittelwert. Mit diesem Mittelwert wird dann der entsprechende Vektor-Index initialisiert.

Dieser Vektor wurde auch mit Matlab gebildet und mit diesen, die in C++ implementiert wurde verglichen, die beide sind identisch. Die Ergebnisse sind im Kapitel Test und Bewertung zu finden.

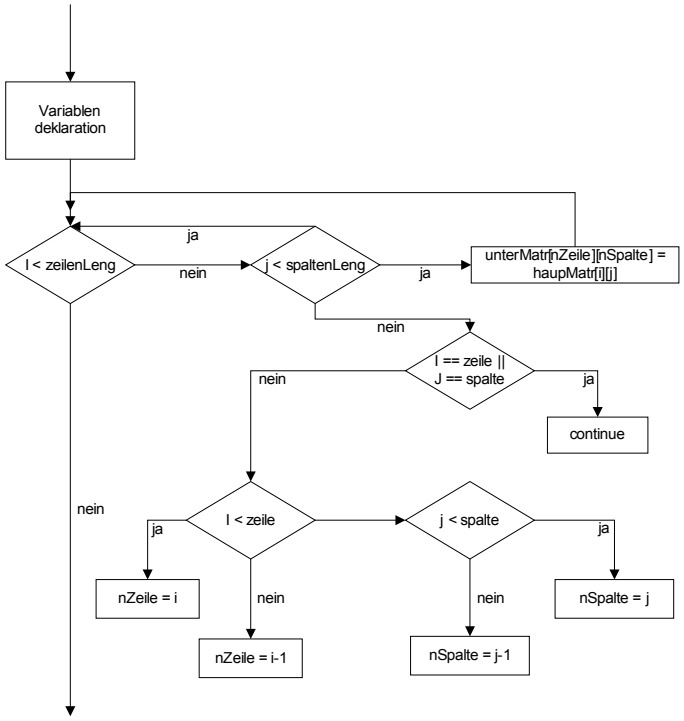


Abbildung 4.15: Implementation der Untermatrix

Damit beim Berechnen der inverse Matrix auch die Determinante der entsprechenden Untermatrix gebildet werden kann muss erst die Untermatrix gebildet werden. Diese Untermatrix ist auch auf dem Heap alloziert, sie wird nur jedes mal mit Werten neu initialisiert. Mit der Funktion $unterMatrix()$ (siehe Abbildung 4.15) der Klasse Parabel wird die Untermatrix mit Werte neu initialisiert. Diese Funktion bekommt als Parameter zwei Integer Werte, der erste ist für die Zeilennummer der zweite ist für die Spaltennummer. In zwei for Schleifen

wird die Hauptmatrix traversiert. Wenn laufende Werte gleich den übergebenen Parametern sind dann wird nichts gemacht. Wenn die laufende Zeile kleiner ist als der übergebene Parameter-Zeile, wird die laufende Zeilenindex als Zeilenindex für die Untermatrix ausgewählt, sonst wird der laufende Zeilenindex mit 1 decrementiert. Das selbe Überprüfung wird auch für der laufende Spaltenindex durchgeführt. Noch innerhalb der beiden for Schleifen wird die Stelle den ausgewählte laufende Zeilen- und Spalten-Index der Untermatrix mit den entsprechende for Schleifen Indexen mit der Wert von der Hauptmatrix initialisiert. Diese Prozedur wird so lange fortgesetzt bis beide for Schleifen traversiert sind.

Es wird erst die Untermatrix in Matlab gebildet und dann in C++ implementiert.

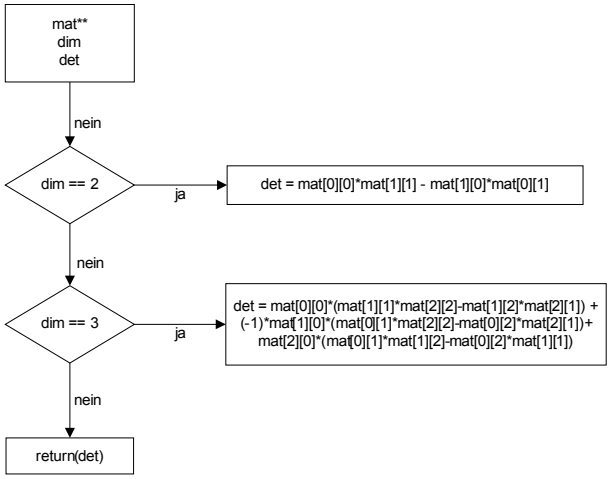


Abbildung 4.16: Implementation der Determinante

Mit der Funktion $berhderminante()$ (siehe Abbildung 4.16) der Klasse Parabel wird die Determinante der als doppelte Pointer übergebenden Matrix berechnet, als zweite Parameter bekommt diese Funktion ein Integer Zahl, die der Rang der Matrix entspricht. Wenn die Matrix der Rang zwei hat dann wird die Regel von Sarrus benutzt, um die Determinante der Matrix zu berechnen. Hat die Matrix den Rang drei, dann wird der Laplacesche Entwicklungssatz benutzt. Diese kann auch sehr gut als eine Rekursion Funktion aufgebaut werden, hier braucht man das aber nicht, den die größte Matrix hat der Rang drei. Der Laplacescher Entwicklungssatz kann in (Hans-Jochen, 2001) gesehen werden, dem Taschenbuch der Mathematik von Bartsch.

Erst wurde die Determionante in Matlab gebildet und dann in C++ implementiert.

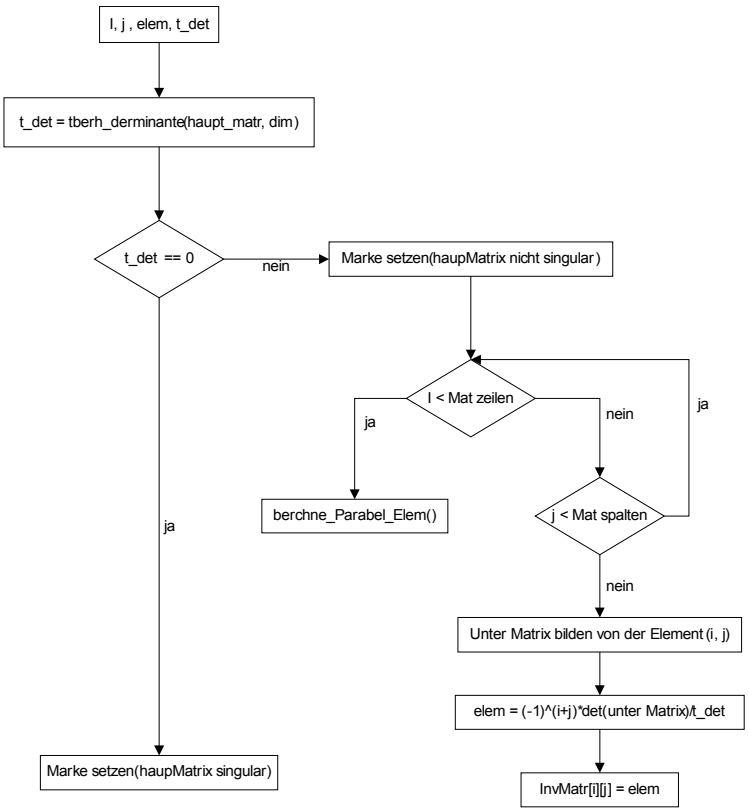


Abbildung 4.17: Implementation der inverse Matrix

Die Funktion $setInverce()$ (siehe Abbildung 4.17) der Klasse Parabel ist eine parameterlose Funktion. Mit dieser Funktion wird die inverse Matrix der Hauptmatrix gebildet. Erst wird die Determinante der Hauptmatrix gebildet. Es wird überprüft ob diese Determinante nicht Null ist. Wegen der Normierung von (mm in m) werden die Werte tausend mal kleiner und am Anfang der Arrays bis zum vierten Index ist der Wert der Determinante sehr klein nah an Null. Aber mit Vergrößerung des laufenden Indexes der Koordinaten-Arrays steigt die Determinante sehr steil fast „exponential“. Die Test sind im Kapitel „Test und Bewertung“ zu sehen. Die Stabilität dieses Algorithus hängt von dem Wert der Determinante ab je größer um so besser. Der Algorithmus besteht aus zwei verschachtelten for Schleifen im Innere der zweiten for Schleife wird die Untermatrix an der entsprechenden Zeile und Spalte Index der Hauptmatrix gebildet. Die Determinante dieser Untermatrix wird gebildet und durch die Determinante der Hauptmatrix dividiert. Der Resultat diese Division wird mit der Potenz der Summe der Zeilen- Spalten-Index der Zahl(-1) multipliziert. So mit jeder diese Ergebnisse wird dann die inverse Matrix im entsprechenden Zeilen- Spalten-Index initialisiert.

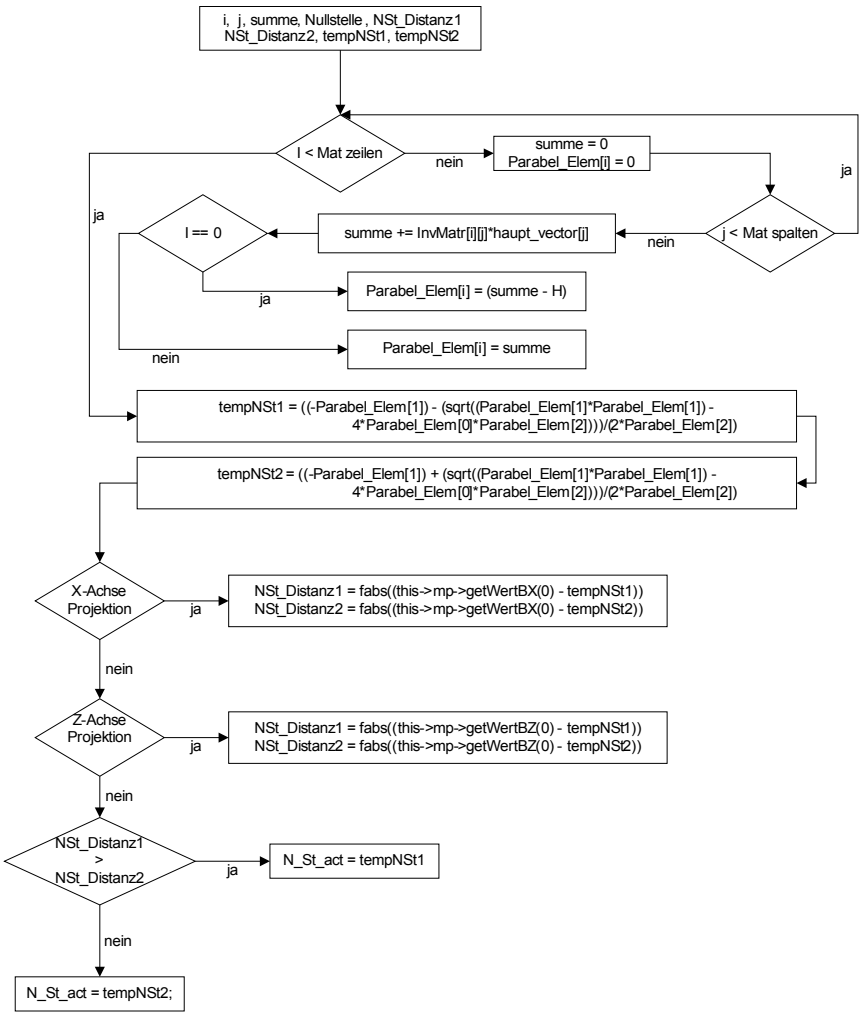


Abbildung 4.18: Implementation der Nullstellen Berechnung

Mit der Funktion $berchneParabelElem()$ (siehe Abbildung 4.18) der Klasse Parabel werden die Nullstellen berechnet und die richtige von beiden ausgewählt. In zwei for Schleifen wird die inverse Matrix traversiert und mit dem transponierten Hauptvektor multipliziert. Die laufenden Multiplikationergebnisse werden aufsummiert. Mit dem Ergebnis am Ende der inneren for Schleife wird dann die dreistellige Array $ParabelElem$ an Stelle des laufenden Zeilen-Indexes initialisiert. Um die Nullstellen zu berechnen, wird die vollständige analytische Formel nicht p,q Formel angewendet, weil dafür müßte durch erste Element der quadratische Gleichung alle Parabel-Koeffizienten geteilt werden. Diese Rechneroperation ist gespart geblieben. Die beide Nullstellen werden berechnet. Erfolgt die Regression in X-Y Projektionsebene, wird dann die richtige Nullstelle durch der Distanz zwischen der Nullstellen und der erste X-Ball-Koordinatenwert ausgewählt. Diese Distanz wird berechnet

und dann mit einander verglichen. Die Nullstelle mit der größere Distanz ist die gesuchte Nullstelle. Erfolgt der Regression in Z-Y Projektionsebene wird der Distanz zwischen den Nullstellen und der erste Z-Ball-Koordinaten Wert berechnet. Die Nullstelle mit der größere Distanz ist die gesuchte Nullstelle.

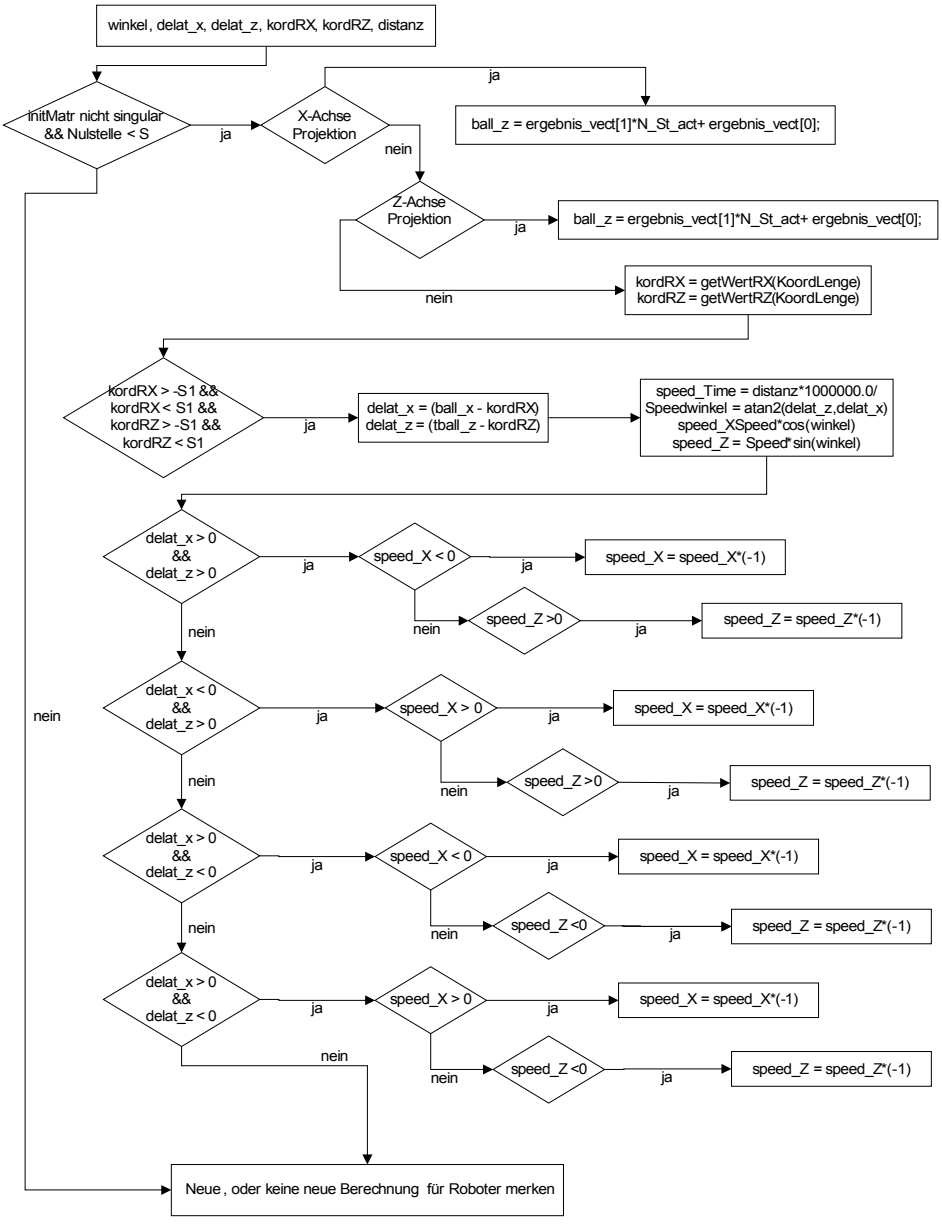


Abbildung 4.19: Implementation der X-Z Koordinate und Roboter Parametern

So wie in der Klasse $Parabel$ wird auch hier erst eine lineare Regression berechnet. Ge-

nau nach dem selben Algorithmus wie in der Klasse Parabel werden die Matrix hier eine 2x2 Matrix und ein Vektor mit der Länge zwei initialisiert. Der Wert der inverse Matrix berechnet und die Koeffizienten der Gerade werden auch berechnet. Diese Gerade ist die Projektion der Parabel auf der X-Z Projektionsebene(Fußboden). Nachdem die Koeffizienten dieser Gerade bekannt sind kann auch der Z-Komponente der Ballslandekoordinate berechnet werden und dann die Robotergeschwindigkeit und Roboterfahrzeit berechnet werden. Hier macht der Entwurf eine kleine Abweichung von der Regelungs-technisches Plan (siehe 4.2).

Es folgt ein Algorithmus zur Berechnung der Z Komponente der Ballslandekoordinate und Roboter-Geschwindigkeit und Roboter-Fahrzeit (siehe Abbildung 4.19). Erst wird berechnet, ob die Nullstelle eine reguläre Nullstelle ist, ob diese außer dem Trackingvolume des ART-Systems geht, und ob die Matrix nicht singulär ist, ist dies der Fall, wird nicht weiter berechnet und als Ergebnis wird eine Marke gesetzt die besagt das keine neue Robotergeschwindigkeit und Roboterfahrzeit berechnet sind. Das bedeutet dass der Roboter weiter mit der „alten“ Geschwindigkeit und Fahrzeit fahren soll. Wenn aber die Matrix nicht singulär ist oder die Nullstelle regulär ist, dann wird folgende Berechnung durchgeführt. Erst wird geprüft in welcher Projektionsebene die Regression durchgeführt wurde. Wenn auf X-Y Projektionsebene wird die Geradengleichung nach Z aufgelöst sonst nach X aufgelöst. Auf diese weise wird auch der Z-Komponente der Ballslandepunkt bekannt. Jetzt sind die Koordinate der Ballslandepunkt bekannt. Die aktuellen Koordinaten des Roboters werden aus der Klasse CUDPDaten ausgelesen. Durch zwei Punkte kann nur eine Gerade durchlaufen (siehe Abbildung 3.6). Es ist zu sehen, dass nach Pythagoras die Distanz zwischen Roboter und Ball berechnet werden kann. Die Geschwindigkeit wird bei Initialisierung mit der Klasse IniReader festgelegt. Die Fahrzeit ergibt sich aus der Distanz dividiert durch die Geschwindigkeit. Um die Geschwindigkeit auf Z und X Richtung zu berechnen, muss erst der Winkel α wie in der Abbildung 3.6 berechnet werden. Der Winkel wird mit der Funktion $atan2()$ der C++ Bibliothek $cmath$ berechnet. Diese Funktion berechnet von 0 bis 180 grad. Die Geschwindigkeit auf X-Richtung ist „Speed*cos((float)winkel)“ die Geschwindigkeit auf Z-Richtung ist „Speed*sin((float)winkel)“. Die Roboter-Geschwindigkeit und die Roboter-Fahrzeit ist jetzt bekannt, aber der Roboter hat eigenes Koordinatensystem (siehe Abbildung 3.13), deshalb muss die Geschwindigkeit auf Z-Richtung einmal negiert werden. Es sind vier Fälle zu unterscheiden. Am ende der Funktion wird einen Flag um gesetzt zu markieren ob es eine neue Berechnung(Robotergeschwindigkeit und Roboterfahrzeit) gibt oder nicht. Dieser Flag wird bei der Steuerung der Roboter benutzt.

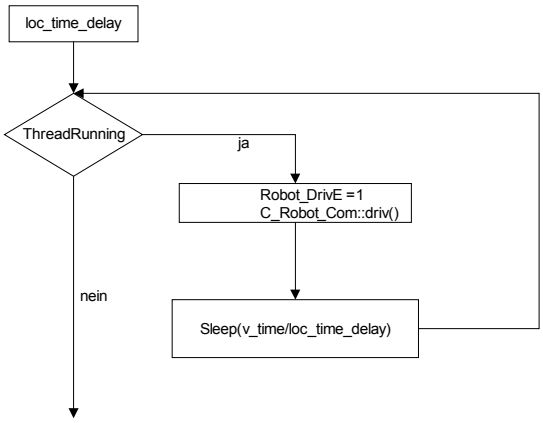


Abbildung 4.20: Implementation der nebenläufige Roboter Funktion

Die nebenläufige Funktion für den Roboter wird durch die Klasse $CMThread$ realisiert. In der Klasse $CRobotCom$ wird die abstrakte Funktion $execute()$ (siehe Abbildung 4.20) implementiert. Diese ist durch eine for Schleife implementiert die immer für einige zehnte Millisekunden durch Windows Funktion Sleep() unterbrochen wird. Als Parameter bekommt Sleep() die gerechnete Fahrzeit dividiert durch eine empirische gefundene Konstante. Wenn die Schleife aus der Sleep-Zustand aufwacht, wird bei neuem Durchlauf der Funktion $drive()$ Aufgerufen. Diese Funktion läuft im Hintergrund bis ende das Programm. Die Endloses „while“ Schleife wird so lange durchgeführt bis die bool Variable die als Besiegung in „while“ Schleife eingesetzt ist mit „false“ inizialisiert wird. Die Besiegung-Variable wird in der Funktion $parseFrame$ der Klasse $CTrack$ mit „false“ Inizialisiert wenn der Ball die unter Wurfhöhe Marke unterschreitet.

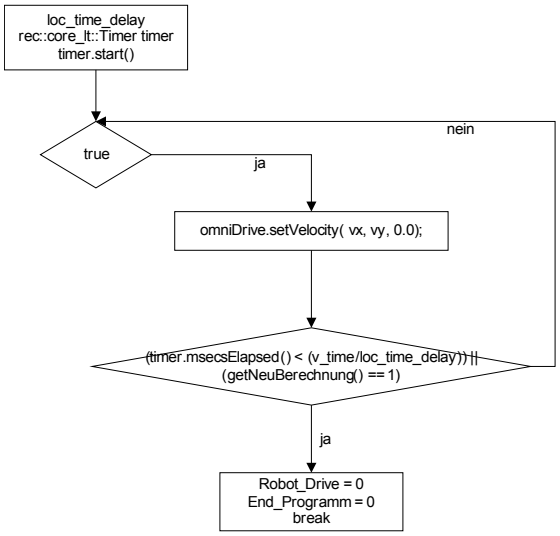


Abbildung 4.21: Implementation der Roboter Drive Funktion

Die Funktion $drive()$ (siehe Abbildung 4.21) der Klasse $CRobotCom$ ist auch mit einer endloses Schleife implementiert. In dieser Schleife wird die Robotino-API Funktion $setVelocity()$ aufgerufen. Diese Funktion erwartet drei Parametern Geschwindigkeit auf X- und Z-Richtung der dritte Parameter ist Winkelgeschwindigkeit, steht immer auf NULL, da hier nicht gebaucht wird. In der Funktion $drive()$ der Klasse $CRobotCom$ wird jedes mal ein Timer gestartet, wenn die Zeit größer ist als die berechnete Roboter-Fahrzeit dividiert(die als Parameter an diese Funktion übergeben wurde) durch die empirische gefundene Konstante oder wenn eine neue Berechnung(neue Geschwindigkeit und neue Fahrzeit) gibt, dann wird diese endlose Schleife verlassen. Bei Verlassen der Schleife wird eine Marke gesetzt das der Roboter „nicht mehr fährt“ damit in diese Zeit seine aktuelle Koordinaten ausgelesen werden könne. Der Roboter sagt auf diese Weise ich stehe für sehr kurze Zeit sieh wo ich bin fahre ich richtig oder nicht. So wird der Roboter immer in die richtige Richtung gebracht, wenn er davon abweicht.

5 Test und Bewertung

5.1 Teststand Aufbauen

Der Ball der in dieser Diplomarbeit benutzt wurde ist aus einem nicht so stabilen Kunststoff hergestellt. Dadurch zerbricht der Ball schnell, wenn er bei einen mißglückten Fangversuch auf den Boden fällt. Aus diesem Grund wurde folgende Testeinrichtung, wie in Abbildung 5.1 zu sehen ist, aufgebaut.

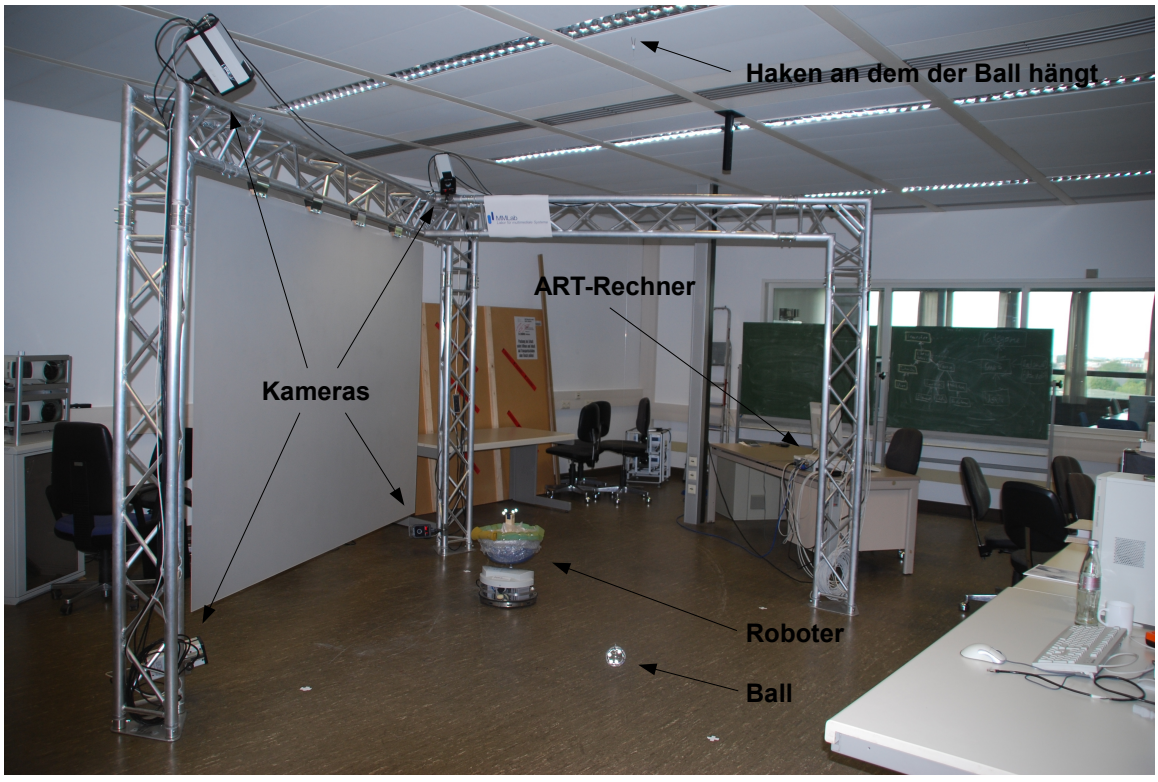


Abbildung 5.1: Teststand

Um zu verhindern das der Ball auf den Boden aufschlägt und zerbricht, ist der Ball mit einem Faden an der Decke befestigt. Wird der Ball vom Roboter gefangen, kann der Ball

durch den Fangkorp beschädigt werden. Aus diesem Grund ist der Fangkorp gepolstert. Der Faden beeinflusst die Flugbahn des Balls so, dass nicht immer auf der X-Z Projktionsebene eine Gerade zu erkennen ist. Diese Abweichung der Geraden tritt eine Abweichung in der Berechnung der Balllandekoordinaten auf. Dieser Fehler wird durch die Größe des Korbs kompensiert.

Die folgende Abbildungen zeigen die Projektionen bei Benutzung dieser Testeinrichtung, bei einem sehr schlechten Wurf des Balls.

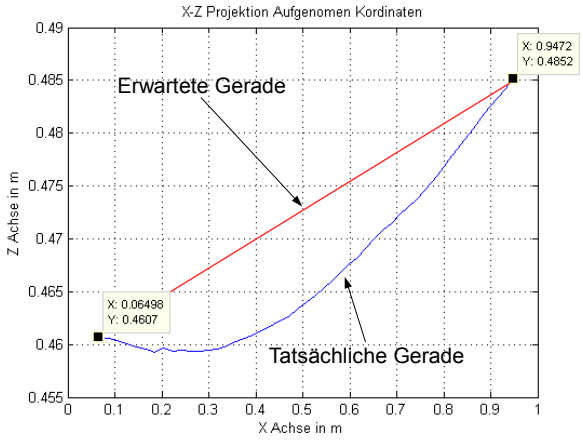


Abbildung 5.2: X-Z Projektion des Ballwurfes mit der Testeinrichtung

Auf der Abbildung 5.2 ist die Projktion der Wurfparabel auf die X-Z Projktionsebene zu sehen. In rot ist die erwartete Projektion (Gerade) und in blau die tatsächliche Projektion zu sehen. Auf der Abbildung ist zu sehen, dass die erwartete Gerade auf der X-Z Projktinsebene nicht zutrifft. Diese Abweichung ist zu erklären, dass der Faden die Ballbahn beeinflusst. Diese Abweichung trifft nicht bei allen Würfen ein, aber bei geübter Wurftechniken tritt diese Abweichung seltener auf.

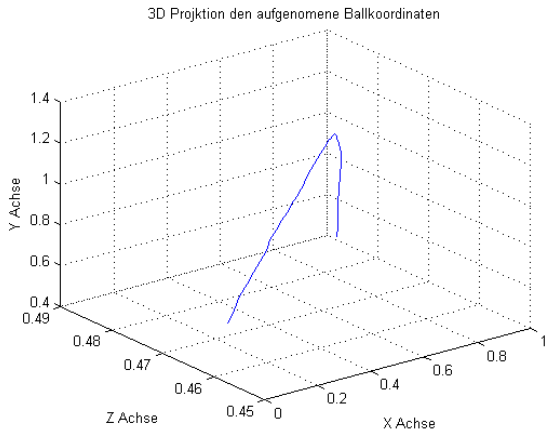


Abbildung 5.3: drei-D Abbildung der Bahlwurf

In der Abbildung 5.3 ist die drei-D Projektion der X-Z Projektion von Abbildung 5.2 zu sehen. Auf dieser Abbildung ist der Einfluss des Fadens auf die Ballbahn nicht stark zu erkennen.

5.2 Testen des Programms

Der Funktionstest wurde wie im vorigen Kapitel beschrieben durchgeführt. In Abbildung 5.5 ist die Projektion des Ballwurfes auf der X-Z(Fußboden)-Projektionseben dargestellt (grüner Graph). Die Abwurfstelle hat die Koordinaten(X=0.5174 m, Z=0.5019 m) und die Ball-Landepunkt hat die Koordinaten (X=1.346 m, Z=0.6667 m). Auf Grund des Einflusses der Angelschnur auf die Ballbahn, ist die Parabel-Projektion auf der X-Z Projektionsebene keine Gerade. Der Roboter hat die Koordinaten(X=1.153 m, Z=0.3554 m) bei dem Abwurf des Balls. Er wird in Bewegung gesetzt, sobald bei der Berechnung eine akzeptable Robotergeschwindigkeit und Roboterfahrzeit als Ergebnis errechnet wird. In diesem Test werden die erste „gute“ Roboter-Geschwindigkeit und-Fahrzeit nach 188 ms (siehe Abbildung 5.5) berechnet, was etwa 23% von der gesammten Flugzeit ist. der Roboter bewegt sich fast geradelinig zu dem errechnete Ball-Landepunkt und seine Endposition hat die Koordinaten (X=1.423 m, Z=0.6852 m). Die Entfernung zwischen Ball-Landepunkt und Endposition des Roboters ist 0.07919 m. Weil der Fangkorb etwa drei mal größer ist als der Ball, ist diese Entfernung tolerierbar. Der Ball wird gefangen.

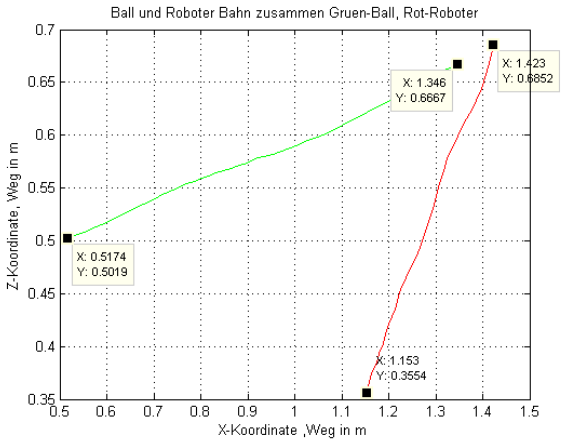


Abbildung 5.4: Roboter und Ball Projektion bei erfolgreicher Test

Es folgt das Geschwindigkeits-Zeit Diagramm des Roboters. Nachdem die erste akzeptable Robotergeschwindigkeit und Roboterfahrzeit ausgerechnet ist, fährt der Roboter mit eine durchschnittlichen Beschleunigung von $4.44\,\frac{\mathrm{m}}{\mathrm{s}^2}$ in seine erste Phase. Die durchschnittliche Geschwindigkeit in der ersten Phase ist $0.3941\,\frac{\mathrm{m}}{\mathrm{s}}$, in der zweiten Phase $0.853\,\frac{\mathrm{m}}{\mathrm{s}}$ und in der dritten Phase $1.023\,\frac{\mathrm{m}}{\mathrm{s}}$. Dies ergibt eine durchschnittliche Geschwindigkeit während des gesamten Fahrwegs von $0.823\,\frac{\mathrm{m}}{\mathrm{s}}$. Bei einer Fahrzeit von $0.531\,\mathrm{s}$ ergibt das einen zurückgelegten Weg von $0.437\,\mathrm{m}$. Diese Distanz, die der Roboter rechnerisch zurückgelegt haben soll, stimmt fast überein mit dem zurückgelegten Weg aus Abbildung 5.4. In dieser Abbildung ist der vom Roboter zurückgelegte Weg nach Pythagoras leicht zu berechnen, denn hier fährt der Roboter fast geradeaus. Der zurückgelegte Weg nach Pythagoras ist $0.426\,\mathrm{m}$. Der Fehler beträgt gerade einmal $0.0101\,\mathrm{m}$.

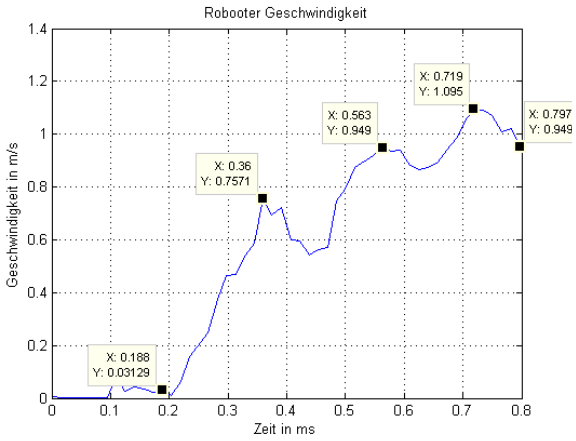


Abbildung 5.5: Roboter Geschwindigkeit bei erfolgreicher Test

Die dazugehörigen Geschwindigkeiten mit dem Systemzeiten: A.23, sind im Anhang zu finden.

Die Abweichung zwischen der berechneten und der realen Nullstelle der Ballparabel A.24, sind im Anhang zu finden.

5.3 Bewertung des Programms

Das Programm funktioniert unter folgende Bedienungen:

- Das ART-System muss sehr gut für dieses Experiment kalibriert werden, damit der Ball und der Roboter immer getrackt werden.
- Der Ball muss nah-parabelförmig innerhalb des Trackingvolumens geworfen werden.
- Der Roboter muss innerhalb des Trackingvolumens sich befinden und vom ART-System auch erfasst sein.
- Der Roboter darf sich nicht weit Entfernt als 0.6 m von dem vermutlichen Balllandepunkt befinden.
- Der Roboter muss entsprechend ausgerichtet sein, bevor der Ball geworfen wird.
- Die Batterie des Roboter muss Voll sein, damit er die geforderte Geschwindigkeit erreichen kann.
- Der Ball darf nicht beschädigt sein, darf nicht stark zerkratzt sein oder an einige Stellen sogar gebrochen.
- Der Fußboden auf dem der Roboter fährt muss trocken sein und nicht schmutzig sonst drehen die Räder durch.

Diese Beschränkungen sind Anhand den Test mit dem Programm entstanden.

5.4 Neue Methode zur Berechnung des Ballaufprallpunkt

Hier wird eine verbesserte Methode zur Berechnung des Balllandepunkt vorgestellt. Diese Methode wird punktförmig beschrieben. Die aktuelle Koordinate des Balls werden mit X_i, Y_i, Z_i bezeichnet. Das Ziel dieser Methode ist es eine eigene Projektionsebene zu bilden, in der das Regressionsverfahren durchgeführt wird. Diese Methode basiert auf eine Hauptkomponentenanalyse(PCA), siehe (Kessler) und auch (Dunteman, 1989).

Die Idee ist, der größte Eigenvektor zum größte Eigenwert der Korrelationsmatrix(wenn diese Matrix aus Messwerte entsteht) zu berechnen. Dieser Vektor zeigt immer entlang der Messreihe. In diesem konkreten Beispiel kann dieser als Basis-Vektor der Projektionsebene benutzt werden. Der zweite Vektor der für Bildung eines orthogonalen Vektorraums ist die Y-Achse mit dem folgenden Wert, $Y = 0, 1, 0$. In diesem neue Vektorraum wird die Polynom-Regression durchgeführt. Und die Koordinaten des Balllandepunkt werden ohne großen zusätzlichem Rechenaufwand Berechnet.

Diese Methode wurde schon in Matlab entwickelt und getestet. Für die Implementierung und Erprobung in C++ fehlt leider die Zeit.

Koordinaten-Vektor-1 des Balls:

$$V_1 = \begin{pmatrix} X_i \\ Z_i \end{pmatrix} \tag{5.1}$$

Koordinaten-Vektor-1 des Balls, transponiert:

$$V_2 = \begin{pmatrix} X_i, Z_i \end{pmatrix} \tag{5.2}$$

- Erst werden die Ballkoordianten(X_i, Y_i, Z_i) auf den Ursprung des ART-Koordinatensystems verlegt(transformiert). Dazu werden jedem Koordinatenwert X_i, Y_i, Z_i der zugehörige Mittelwert abgezogen. Die neuen transformierten Ballkoordinaten sind dann X_{ni}, Y_{ni}, Z_{ni}.
- Von den Vektoren V_1 und V_2 wird eine 2xn-Matrix erzeugt.
- 2xn-Matrix wird transponiert.
- Es wird die 2xn mit der transponierten 2xn-Matrix multipliziert. So wird die 2x2-Korrelationsmatrix gebildet.
- Die Eigenwerte der Korrelationsmatrix werden berechnet und der größte ausgewählt.
- Zu dem größten Eigenwert wird der größte Eigenvektor berechnet, er wird als E_g bezeichnet.
- Der Eigenvektor E_g wird von 2D-Vektor zu ein 3D-Vektor umgewandelt, indem in der Mitte des E_g ein 0 geschrieben wird.

  Danach sieht der Eigenvektor so aus: $E_g = (Wert_0, 0, Wert_1)$ und wird E_{gn} bezeichnet. Der zweiter Vektor für die neue Projektionsebene ist die Y-Achse der ART-System. Diese Y-Vektor hat den folgender Wert $Y = (0, 1, 0)$.

- Die transformierten Ballkoordinaten X_{ni}, Y_{ni}, Z_{ni} werden einmal mit E_{gn} und einmal mit dem Y-Vektor multipliziert(Skalarprodukt von zwei Vektoren). Das Ergebnis sind zwei Vektoren die die neue 2D-Projektionsebene bilden. Jetzt kann das Regressionsverfahren in diese neuen Projektionsebene durchgeführt werden. Das Ergebnis des Regressionsverfahren sind zwei Nullstellen Nst_1 und Nst_2.
- Die berechneten Nullstellen werden mit dem E_{gn} multipliziert so werden die Nullstellen-Vektoren Nsa_{1Eg} und Nsa_{2Eg} bestimmt. Beide Vektoren enthalten die Landekoordinaten des Balls. Um die richtige Nullstelle zu berechnen, werden Differenzen zu den echten Ballkoordinaten gebildet: $R = ((X_n - X_0), (Y_n - Y_0), (Z_n - Z_0))$.
- Die Nullstellen-Vektoren werden mit dem R-Vektor multipliziert. Das Ergebnis sind zwei Werte N1 und N2.
- Beide Werte N1 und N2 werden verglichen. Wenn (N1 > N2) ist, dann ist Nsa_{1Eg} der richtige Nullstellen-Vektor wenn (N1 < N2) dann ist der Nsa_{2Eg} der richtige Nullstellen-Vektor.

Mit einem Testwurf wurden folgende Koordinaten in Matlab berechnet:

$$\begin{pmatrix} x = 1.2545 \\ y = 0.0508 \\ z = 0.8763 \end{pmatrix} \tag{5.3}$$

Im der Abbildung A.25 sind die Koordinaten mit dem Index = 55 die Balllandekoordinaten des Balls zu sehen.

$$\begin{pmatrix} x = 1.26222 \\ y = 0.667612 \\ z = 0.861991 \end{pmatrix} \tag{5.4}$$

- Differenz bei X-Werte = $0.00677\ m$, es ist eine Abweichung von $6\ mm\%$
- Differenz bei Z-Werte = $0.0143\ m$, es ist eine Abweichung von $1.4\ cm\%$

Diese Abweichung kommt zustande, da der Flug des Balls nicht bis zur Höhe des Roboterfangkorb aufgezeichnet wurde. Die Abweichung ist aber akzeptabel. Diese Berechnung ist mit 100% der Messwerte durchgeführt worden. Damit der Roboter den Ball fangen kann, müssen die Koordinaten des Aufschlagpunkt des Balls so früh wie möglich berechnet werden. Die Abweichung zwischen tatsächlichem Landepunkt und berechneten Landepunkt muss akzeptabel sein.

Die folgende Aufzählung zeigen die berechneten Koordinaten des Landepunktes bei unterschiedlichen Fortschritt der Wurfparabel an:

- bei 25 % der Koordinatenpunkten , (1.6370 0.508 0.2733).
- bei 35% der Koordinatenpunkten , (1.3382 0.508 0.7251).
- bei 40% der Koordinatenpunkten , (1.3059 0.508 0.7770).
- bei 45% der Koordinatenpunkten , (1.2771 0.508 0.8251).
- bei 50% der Koordinatenpunkten , (1.2613 0.508 0.8531).

Die Differenz zwischen tatsächlichen Landepunkt-Koordinaten und den Berechneten wird kontinuierlich kleiner. Bei 50% ist die Abweichung auf der X-Achse 0.9 mmm und auf Z-Achse 8 mm. Diese Abweichungen sind akzeptabel und mit diesem Algorithmus kann auch gearbeitet werden.

Die folgenden Abbildungen beziehen sich auf dem Ballwurf mit den Koordinatentabelle (siehe Abbildung A.25) im Anhang. In der Abbildung 5.6 ist die Projektion der Ballbahn auf X-Z Projektionsebene dargestellt.

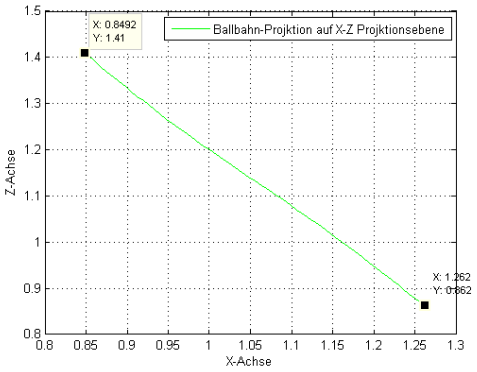


Abbildung 5.6: X Z Projektionen des Balls

Nachdem die Koordinaten des Balls auf der X-Z Projektionsebene in der Ursprung des Koordinatensystems verlegt sind, (siehe Abbildung 5.7), so wie in hier beschriebenen Al-

gorithmus ist. Der größte Eigenvektor zu dem größte Eigenwert zeigt im Abbildung 5.7 entlang der verlegte Messwerte.

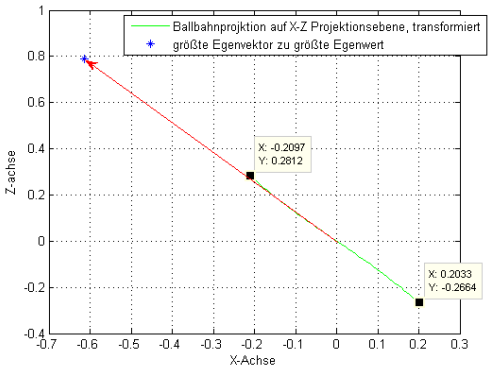


Abbildung 5.7: X Z Projektionen des Balls transformiert

6 Zusammenfassung und Ausblick

Die Zielsetzung der vorliegenden Arbeit ist, einen geworfenen Ball in einer vorher bekannten Testumgebung zu werfen und diesen noch in der Luft mit einem auf dem Roboter installierten Fangkorb zu fangen.

Für diese Arbeit wurden drei Methoden entworfen. Bei der erste Methode wird eine Projektionsebene festgelegt. Hierbei wird die Wurfparabel durch Translation und Drehung, mittels Drehmatrizen, parallel zur ausgewählten Projektionsebene ausgerichtet. Es werden vier Fälle bezüglich der Winkel zwischen dem Parabelwurf und der Projektionsebene berücksichtigt. Der Rechenaufwand steigt hier mit der Zahl der Messdaten durch die ständige Ausrichtung des Parabelwurfes. Diese Methode ist die rechenintensivste der drei vorgestellten. Es kann optimiert werden indem nicht jeder Messpunkt verwendet wird, dabei sinkt allerdings die Genauigkeit.

In der nächsten Methode wird eine Projektionseben aufgebaut. Diese Methode basiert auf dem (PCA) Algorithmus, siehe Kapitel 5.4. Diese ist nur auf Matlab implementiert und getestet worden. Für diese Methode blieb keine Zeit zur Implementierung in der Hardware. Diese Methode ist nicht so Rechenaufwandig wie die erste Methode, allerdings sehr genau. Bei der nächsten Implementation ist diese zu empfehlen durch Ihre hohe Genauigkeit.

Bei der letzten, implementierten Methode wird die Projektionsebene je nach Winkel (Roboterfahrgeschwindigkeit und -zeit) ausgewählt. Diese ist vom Prinzip sehr einfach, dafür aber nicht so genau wie die beiden vorherigen. Der Vorteil bei diese Methode ist die schnelle Berechnung. Der Fehler ist mit der Große des Fangkorb kompensiert und beträgt maximal 0.03 m bei 100 % der Messpunkte wenn unter einem Winkel von 45 ° zu X- oder Z-Achse der Ball geworfen wird.

Um den Schlupf zwischen den Rädern des Roboters und dem Fußboden zu reduzieren, um damit bei bestehender Roboterregelung eine höhere Beschleunigung und Geschwindigkeit zu erzielen, wäre ein Fußbodenbelag mit größerer Bodenhaftung von Vorteil. Dadurch kann eine größere Distanz zwischen Roboter und errechnetem Aufprallpunkt des Balls zurückgelegt werden.

Bei einer erneuten Implementation wäre die zweiten Methode empfehlenswert. Weil diese rechenaufwendiger ist als die jetzige, ist es besser ein Realtime Betriebssystem zu benutzen. So kann bei der Berechnung eine präzise Nebenläufigkeit erzielt werden. Auf diese Art wird Rechenzeit gespart.

Der Ball der in dieser Aufgabe benutzt wurde ist sehr zerbrechlich. Ein Ball der nicht so zerbrechlich ist, wäre von Vorteil. Bei der Zerstörung eines Balls muss ein neuer Body im Ball aufgebaut werden und das ART-System neu auf diesen Body kalibriert werden. Zudem wird die Oberfläche des Balls bei jeden Wurf in Mitleidenschaft gezogen was eine fehlerfrei Durchdringung der Infrarotstrahlen verhindert. Das hat zur Folge das der Ball während des Fluges nicht immer identifiziert wird. Dies erhöht die Fehleranfälligkeit der Berechnung.

Die Kalibrierung des ART-System ist sehr wichtig für dieses Programm. Es hat sich sich herausgestellt das der Ball und der Roboter nicht von dem System während des Ballflugs nicht immer erkannt werden. Das mindert die Genauigkeit bei der Berechnung des Balllandepunktes.

In diese Arbeit wurden MS-System Threads benutzt. Um eine bessere Portierbarkeit auf anderen Betriebssysteme z.B. Linix wäre von von Vorteil die „Boost-C++“Bibliothek zu benutzen. Dafür kann das Buch (Scott) benutzt werden.

Diese Arbeit hat gezeigt, dass es Möglich ist mit dieser Roboter(Robotino) der Firma (Festo), in einer vorher bekannten Tracking Volume die von Firma ART (ART)) aufgebaut ist, ein Ball, wenn dieser parabelförmig geworfen wird noch in der Luft von dieser Robotino zu fangen.

Zukünftige Perspektiven zu Fortsetzung diese Arbeit sind, z.B.:

Im nächsten Schritt sollte das tracking Typ der Testumgebung bis jetzt mit der Firma (ART), durch ein Roboter eigenes „Computer Vision“- System(zwei Kameras auf dem Roboter installieren). Mit diese „Computer Vision“ soll der Roboter autonom in der Lage sein, fliegender Ball zu identifizieren dieser der Landepunkt-Koordinaten zu berechnen und dahin zu fahren bevor der Ball auf dem Boden fehlt.

Im Hinsicht auf Roboter-Handball ist es wünschenswert eine Vorrichtung auf dem Roboter zu Installieren, die den gefangenen Ball auf eine Abwurfvorrichtung bringt, um dann wieder der Ball abwerfen zu können.

Um ein Zusammenspiel mehrere Handball-Roboter zu ermöglichen ist es zusätzlich ein dynamischen Planungssystem zu entworfen. Mit diese soll die Roboter unter sich kommunizieren, damit der bei der Abwurf und fangen des Balls eine Synchronisation den Roboter-Bewegung gibt. Ziel ist es keine Zusammenstoß mehrere Roboter beim fangen des Ball zu verhindern. Die Roboter müssen unter ein andern Kommunizieren damit, wenn einen Roboter der bevor der Ball wirft ein Bestätigung von ein anderen Roboter bekommen, dass er bereit ist der Ball zu fangen. Wenn mehrere Roboter diese Bestätigung absenden muss eine erneuerte Befragung durchgeführt werden bis nur ein einziger antwortet. Alle anderen sollen, erst dann einspringen wenn bei der Berechnung des ersten Roboter einen Fehler auftritt oder die Distanz des Roboters zum voraussichtlichen Ball-Landepunkt zu groß ist. Ist dies der Fall, muss der Roboter, der sich am nächsten am voraussichtlichen Ball-Landepunkt befindet kontaktiert werde, damit der Ball nicht auf dem Boden fällt.

RoboCup ist eine internationale Initiative, die Roboterforschung fördert (Nicholas Adjouri). Der Präsident der International RoboCup Federation ist Professor Hiroaki Kitano und die offizielle Internetseite ist (RoboCup-Federation). Es existieren drei Kategorien: RoboCup-Fußball, RoboCup-Dance und RoboCup-Rescue. In RoboCup-Fußball, wie der Name sagt, treten Roboter als Fußballmannschaften gegeneinander an. In RoboCup-Rescue führen die Roboter Rettungseinsätze durch. Als eine weitere Kategorie kann auch RoboCup-Volleyball eingeführt werden, diese Disziplin wird dynamischer und aufregender als RoboCup-Fußball.

Literaturverzeichnis

[activrobots] ACTIVROBOTS: *Hompage der Company*. – Online verfügbar http://www.activrobots.com/;
besucht im Juli 2009

[Adam 2006] ADAM, Stefan R. A.: *MATLAB und Mathematik kompetent einsetzen*. 1. Auflage - April 2006. Wiley-VCH, Berlin Verlag, 2006

[ART] ART, Avanced Realtime Tracking G.: *Optikaltracking ART System*. – Online verfügbar http://www.ar-tracking.de;
besucht im Uni 2009

[Bednarik] BEDNARIK, Karl: *Laminare Strömunen*. – URL http://de.wikipedia.org/w/index.php?title=Datei:LAMINARB_Laminare_Stroemung.PNG&filetimestamp=20050925104930

[Braun 2005] BRAUN, Anton: *Grundlage der Regelungstechnik*. 1 Auflage. Fachverlag Leipzig, 2005

[Bräunl] BRÄUNL, Thomas: *Embeddet Robotics*. ‘3 Auflage

[Dunteman 1989] DUNTEMAN, George H.: *Principal Component Analysis*. Sage University paper, 1989. – Online verfügbar http://books.google.com/books?id=Pzwt-CMMt4UC&printsec=frontcover&hl=de;
besucht im Juli 2009

[fastlabinc] FASTLABINC: *PracticalSocket.h und C++ File*. – URL http://fastlabinc.com/CSL/doxygen/_practical_socket_8h-source.html

[Festo] FESTO, Festo Didactic G.: *Festo Tehnical support*. – Online verfügbar http://www.festo-didactic.com/de-de/lernsysteme/robotino-forschen-und-lernen-mit-robotern/;
besucht im Juni 2009

[Fujitsu] FUJITSU: *Hompage der Company*. – Online verfügbar http://pr.fujitsu.com/en/news/2001/09/10.html;
besucht im Juli 2009

[Hans-Jochen 2001] HANS-JOCHEN, Bartsch: *Taschenbuch Mathematische Formeln*. 19.,Ausgabe. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2001. – 482,484 S

[HAW-Hamburg] HAW-HAMBURG: *PracticalSocket header und C++ File*. – Online verfügbar `http://cvs.informatik.haw-hamburg.de/cgi/viewvc_cvs.cgi/inb5wd/DW2/DTrack/PracticalSocket.h?view=log`;
besucht im Juni 2009

[H.v.Mangoldt's 1961] H.V.MANGOLDT'S, Konrad K.: *Einführung in die Höhere Mathematik, Zum Studium und selbsstudium*. 11 Auflage. S.Hirzel Verlag Stutgart, 1961

[Isermann 2008] ISERMANN, Rolf: *Mechatronische Systeme*. 2 Auflage. Springer Verlag, 2008

[K-Team] K-TEAM: *Hompage der Company*. – Online verfügbar `http://www.k-team.com/kteam/home.php?rub=0&site=1&version=EN`;
besucht im Juli 2009

[Kessler] KESSLER, Waltraud: *Multivariate Datenanalyse*. Fachverlag Wiley-VCH. – Online verfügbar `http://books.google.com/books?id=gPCscQKh-ekC&pg=PA21&dq=Hauptkomponentenanalyse&hl=de`;
besucht im Juli 2009

[Kölzer] KÖLZER, Haw-Hamburg Hans P.: *Digitale Bildverarbeitung und Mustererkennung*. – Online verfügbar `http://users.etech.haw-hamburg.de/users/Koelzer/ibv.htm`;
besucht im August 2009

[Manfred Reuter] MANFRED REUTER, Serge Z.: *Regelungstechnik für Ingenieure*. 11 Auflage. Vieweg Verlag. – 17,32 S

[Matlab] MATLAB: *MathWorks Deutschland*. – Online verfügbar `http://www.mathworks.de/`;
besucht im Juni 2009

[Meisel 2009] MEISEL, HAW-Hamburg A.: *Bildverarbeitung*. 2009. – Online verfügbar `http://www.informatik.haw-hamburg.de/meisel.html?&0=`;
besucht im Juli 2009

[Nehmzow 2002] NEHMZOW, Ulrich: *Mobile Robotik eine praktische Einführung*. 1 Auflage. Springer Verlag, 2002

[Nicholas Adjouri] NICHOLAS ADJOURI, Petr S.: *Sport-Branding: Mit Sport-Sponsoring zum Markenerfolg*. Gabler. – Online verfügbar `http://books.google.de/books?id=_qEPrL1jUX4C&pg=PA169&dq=robocup#v=onepage&q=robocup&f=false`;
besucht im August 2009

[Peter Prinz 2002] PETER PRINZ, Ulla Kirch-Prinz: *C++ Lernen und professionell anwenden*. 2 Auflage. mitp Fachverlag, 2002

[Renz a] RENZ, HAW-Hamburg W.: *Multimedia Systems, Vorlesungsmaterialien WS 2008/2009*

[Renz b] RENZ, HAW-Hamburg W.: *Numerische Mathematik, Vorlesungsskript 2008*. – Kapitel 5.7, Seite-59

[Richard Hartley 2003] RICHARD HARTLEY, Andrew Z.: *Multiple View Geometry*. 2 Auflage. Cambridge Verlag, 2003

[RoboCup-Federation] ROBOCUP-FEDERATION: *Offizielle Seite RoboCup*. – Online verfügbar `http://www.robocup.org/`;
besucht im August 2009

[Roddeck 2002] RODDECK, Werner: *Einführung in die Mechatronik*. 2 Auflage. Teubner Fachverlag, 2002. – Online verfügbar `http://books.google.com/books?id=Kz5YEtzBNnEC&pg=PR1&dq=einf%C3%BChrung+in+Mechatronik&hl=de`;
besucht im Juli 2009

[Rolf Dieter Schraft 1996] ROLF DIETER SCHRAFT, Hansjörg V.: *Serviceroboter innovative Technik in Dienstleistung und Versorgung*. 1 Auflage. Springer Verlag, 1996

[Rolf Dieter Schraft 2004] ROLF DIETER SCHRAFT, Kai W.: *Service Roboter Visionen*. 1 Auflage. Hanser Verlag, 2004

[Schwald 2006] SCHWALD, Bernd: *Punktbasiertes 3D-Tracking starrer und dynamischer Modelle mit einem Stereokamerasystem für Mixed Reality*. Dissertation TU Darmstadt, 2006. – Online verfügbar `http://tuprints.ulb.tu-darmstadt.de/734/`;
besucht im August 2009

[Scott] SCOTT, Meyers: *Effektiv C++ programmieren*. ‘3 Auflage

[Stroppe 2003] STROPPE, Heribert: *Physik für Natur- und ingenieurwissenschaften*. 12 Auflagee. Fachbuchverlag Leipzig, 2003

[Thomas 2000] THOMAS, Andreae: Mathematik für Studierende der Informatik-2. (2000)

[Wany] WANY, Robotics: *Hompage der Company*. – Online verfügbar `http://www.wanyrobotics.com`;
besucht im Juli 2009

A Anhang

A.1 Vorüberlegungen zum Steuerungsmodell des Roboters

Es folgend Bilder zu Bestimmung der maximalen stabilen Geschwindigkeit des Roboters auf dem Boden im MMLab. Dem Roboter wird eine konstante Geschwindigkeit auf der X- und Z-Achse vorgegeben. In den Abbildungen wird der zurückgelegte Weg des Roboters als eine blaue Linie dargestellt. Diese Linie muss eine Gerade mit einen Winkel von 45° zur Z-Achse sein. Weicht diese Linie von der Geraden ab, ist die maximale stabile Geschwindigkeit überschritten.

A.1.1 Stabilität der Bewegung mit einmaliger Vorgabe der Geschwindigkeit

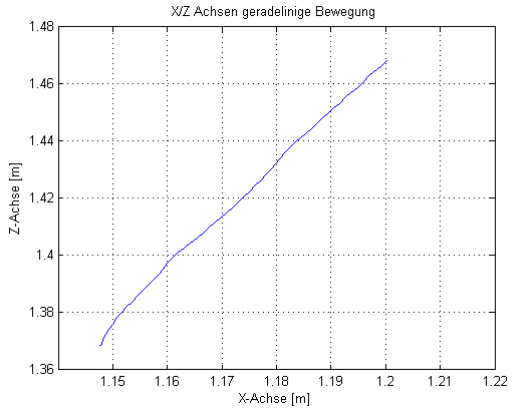


Abbildung A.1: Bewegungsverhalten bei 100 mm/s in einem Winkel von 45° ohne Zeitraster

refernezieren

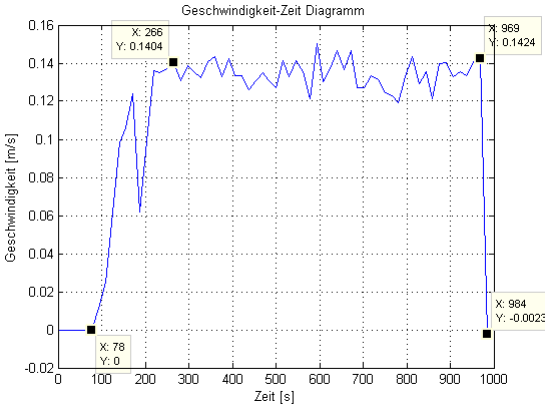


Abbildung A.2: Geschwindigkeit bei 100 mm/s in einem Winkel von 45° ohne Zeitraster

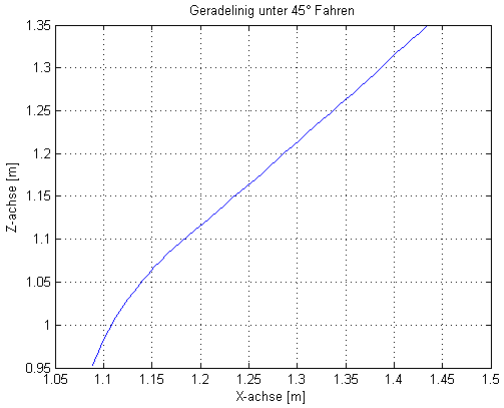


Abbildung A.3: Bewegungsverhalten bei 500 mm/s in einem Winkel von 45° ohne Zeitraster

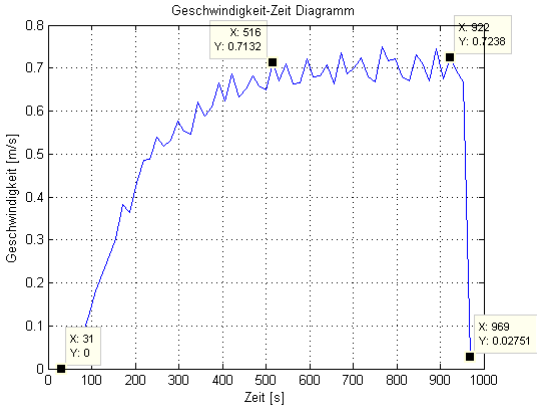


Abbildung A.4: Geschwindigkeit bei 500 mm/s in einem Winkel von 45° ohne Zeitraster

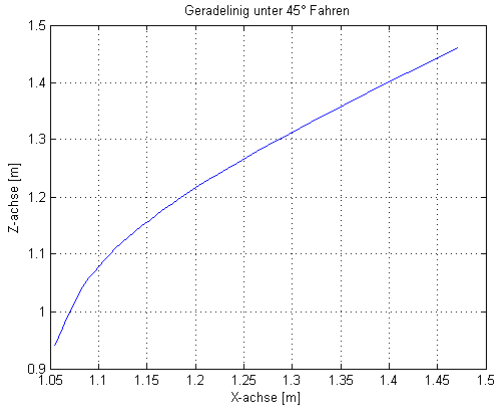


Abbildung A.5: Bewegungsverhalten bei 700 mm/s in einem Winkel von 45° ohne Zeitraster

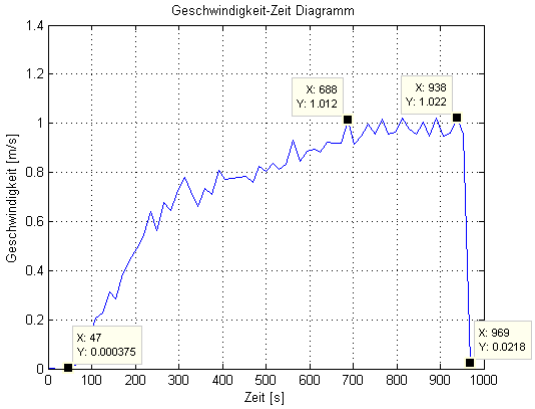


Abbildung A.6: Geschwindigkeit bei 700 mm/s in einem Winkel von 45° ohne Zeitraster

Hier sind die Geschwindigkeit-Zeit Diagrammen und die stabilitäts Diagramme:

Stabilitäts-Diagramm bei eine Geschwindigkeit von 100 mm/s

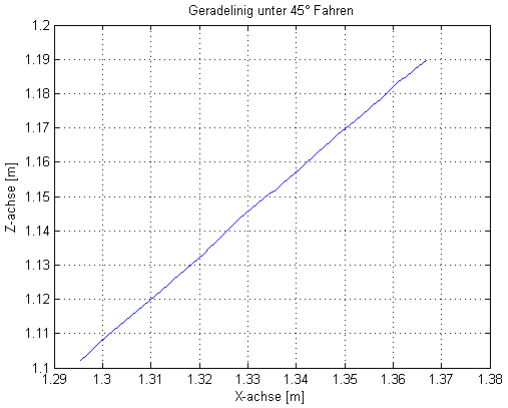


Abbildung A.7: Bewegungsverhalten bei 100 mm/s in einem Winkel von 45° mit Zeitraster

Geschwindigkeit-Zeit-Diagramm bei eine Geschwindigkeit von 100 mm/s

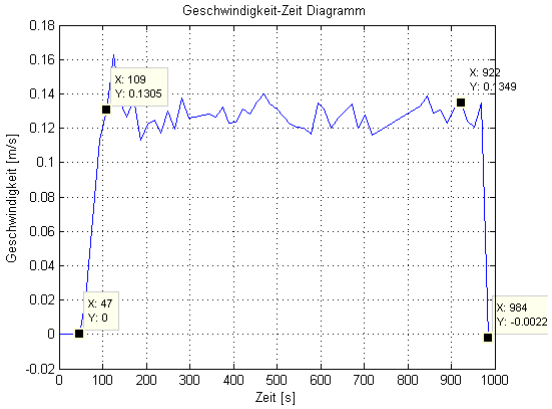


Abbildung A.8: Geschwindigkeit bei 100 mm/s in einem Winkel von 45° mit Zeitraster

Stabilitäts-Diagramm bei eine Geschwindigkeit von 300 mm/s

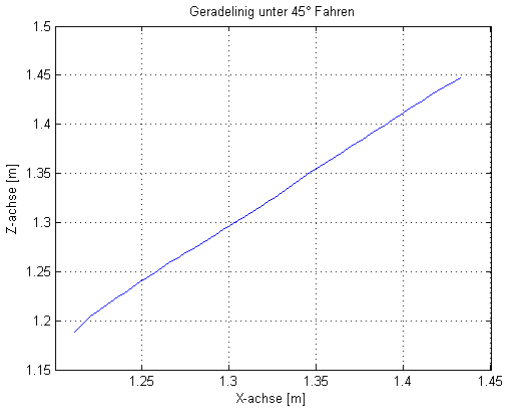


Abbildung A.9: Bewegungsverhalten bei 300 mm/s in einem Winkel von 45° mit Zeitraster

Geschwindigkeit-Zeit-Diagramm bei eine Geschwindigkeit von 300 mm/s

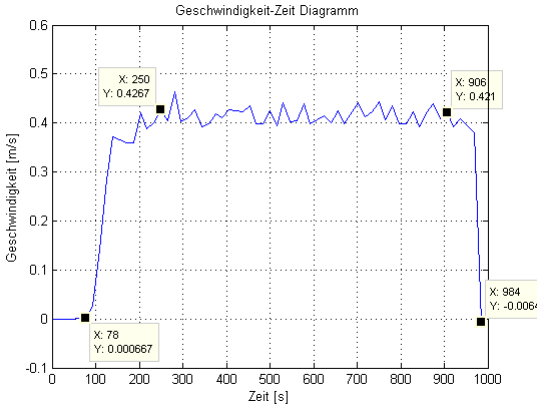


Abbildung A.10: Geschwindigkeit bei 300 mm/s in einem Winkel von 45° mit Zeitraster

Stabilitäts-Diagramm bei eine Geschwindigkeit von 500 mm/s

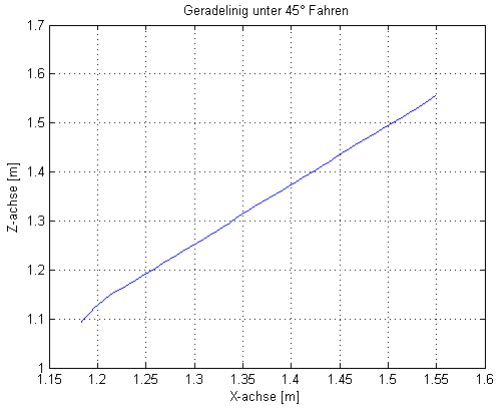


Abbildung A.11: Bewegungsverhalten bei 500 mm/s in einem Winkel von 45° mit Zeitraster

Geschwindigkeit-Zeit-Diagramm bei eine Geschwindigkeit von 500 mm/s

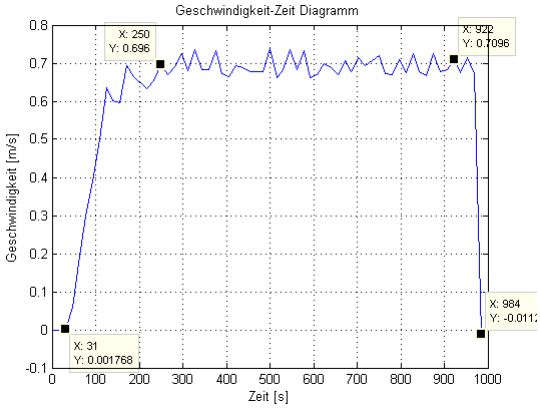


Abbildung A.12: Geschwindigkeit bei 500 mm/s in einem Winkel von 45° mit Zeitraster

Stabilitäts-Diagramm bei eine Geschwindigkeit von 700 mm/s

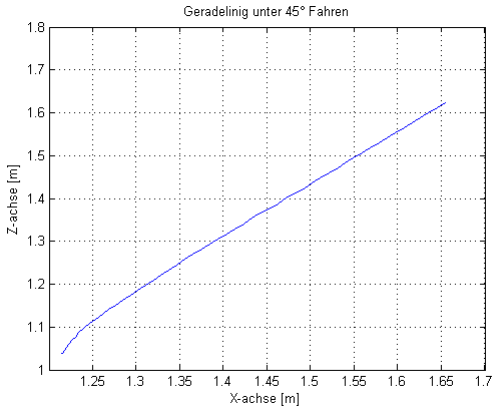


Abbildung A.13: Bewegungsverhalten bei 700 mm/s in einem Winkel von 45° mit Zeitraster

Geschwindigkeit-Zeit-Diagramm bei eine Geschwindigkeit von 700 mm/s

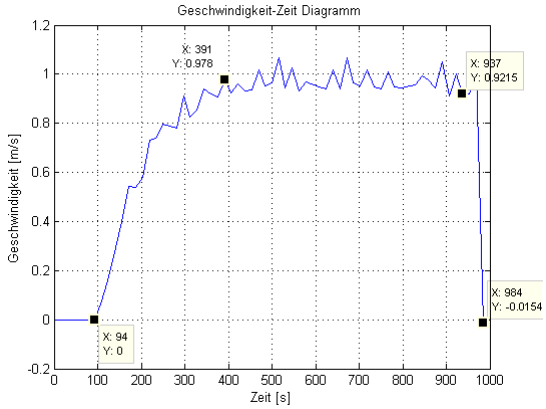


Abbildung A.14: Geschwindigkeit bei 700 mm/s in einem Winkel von 45° mit Zeitraster

Stabilitäts-Diagramm bei eine Geschwindigkeit von 800 mm/s

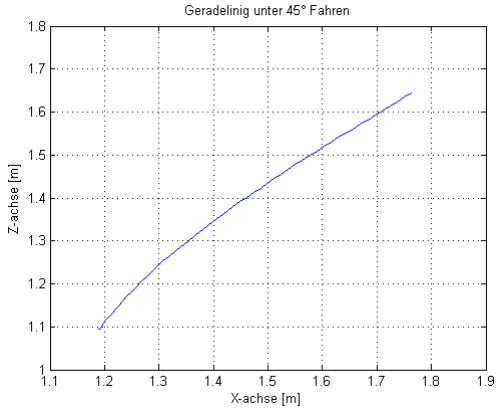


Abbildung A.15: Bewegungsverhalten bei 800 mm/s in einem Winkel von 45° mit Zeitraster

Geschwindigkeit-Zeit-Diagramm bei eine Geschwindigkeit von 800 mm/s

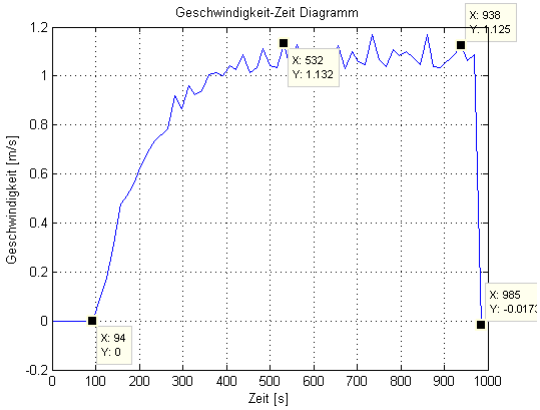


Abbildung A.16: Geschwindigkeit bei 800 mm/s in einem Winkel von 45° mit Zeitraster

Stabilitäts-Diagramm bei eine Geschwindigkeit von 900 mm/s

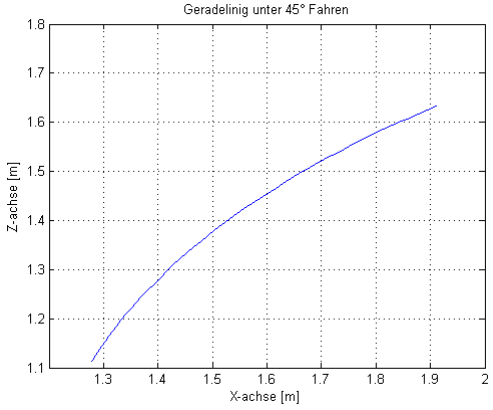


Abbildung A.17: Bewegungsverhalten bei 900 mm/s in einem Winkel von 45° mit Zeitraster

Geschwindigkeit-Zeit-Diagramm bei eine Geschwindigkeit von 900 mm/s

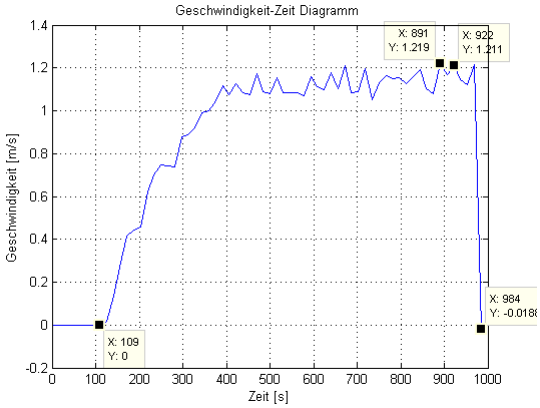


Abbildung A.18: Geschwindigkeit bei 900 mm/s in einem Winkel von 45° mit Zeitraster

A.2 Technische Daten des Roboters

Leistungsdaten Motor der Roboter-Robotino

Gleichstrommotor(GR 42x25)	Einheit	Wert
Nennspannung	V DC	24
Nenndrehzahl	rpm	3600
Nenndrehmoment	Ncm	3.8
Nennstrom	A	0.9
Anlaufmoment	Ncm	20
Anlaufstrom	A	4
Leerlaufdrehzahl	rpm	4200
Leerlaufstrom	A	0.17
Entmagnitisierungsstrom	A	0.17
Trägheitsmoment	gcm2	71
Motorgewicht	g	390

Tabelle A.1: Leistungsdaten des Robotino-Motors

Getriebe der Roboter-Robotino

Planetengetribe(PLG 42 S)	
1-stufug Nm:	3.5
1-stufig i:	4:1-8:1
2-stufig Nm:	6
2-stufig i:	16:1-64:1
3-stufig Nm:	14
3-stufig i:	100:1-512:1

Tabelle A.2: Getriebe der Roboter-Robotino

Alseitenrolen der Roboter-Robotino

Allseitenrolle, angetrieben(ARg 80)	
Durchmesser	80 mm
Maximale Tragfähigkeit	40 kg

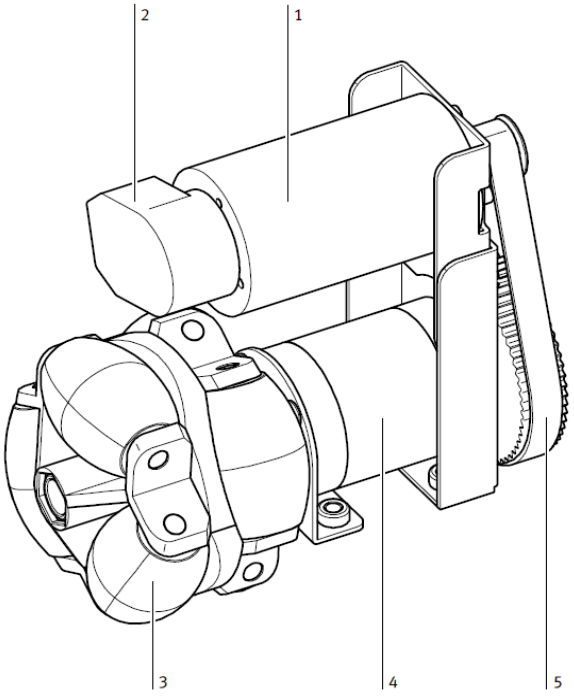
Tabelle A.3: Alseitenrolen der Roboter-Robotino

Der Roboter-Robotino ist mit omnidrive Antrieb System [1] ausgestattet. Diese Antrieb steigert seine Rangiermöglichkeiten enorm, der Roboter braucht um nach links oder unter 45 ° zu Fahren nicht vorher zu drehen.

Der Antrieb des Robotino® geschieht durch 3 unabhängig agierende omnidirektionale Antriebseinheiten. Sie stehen in einem Winkel von 120" zueinander.

Jede dieser Antriebseinheiten besteht aus folgenden Komponenten:

- Gleichstrommotor
- Getriebe mit einer Übersetzung von 16:1
- Allseitenrolle
- Zahnriemen
- Inkrementalgeber



Motor (1) Inkrementalgeber (2) Allseitenrolle (3) Getriebe (4) Zahnriemen (5)

Alle Einzelkomponenten sind auf dem hinteren Befestigungsflansch montiert. Zusammen mit dem vorderen Flansch wird die Antriebseinheit durch Schrauben am Chassis befestigt. So wird die korrekte Lage der Antriebseinheiten zueinander gewährleistet.

Abbildung A.19: Robotino Omnidrive Antrieb (Festo)

[1] Onmidrive Bedeutet in alle Richtungen fahrbar ohne vorher eine drehe Bewegung durchzuführen

Die maximale Geschwindigkeit die der Roboter erreichen kann ist $10\frac{km}{h}$ das entspricht $2,8\frac{m}{s}$, diese Information über die Geschwindigkeit ist bei(Festo) telefonisch zu erfahren. Die Erfahrungen in diese Diplomarbeit zeigen das dies bei idealen Bedienungen(Sehr gute Reibung) möglich ist.

A.3 DGL für Luftwiderstand von Technischealyse

Die DGL 3.8 ist die erste Gleichung einer homogenen DGL mit konstanten Koeffizienten weil sie keine Störfunktion hat. Die DGL 3.9 ist die zweite Gleichung diese ist eine inhomogenen DGL mit Konstanten Koeffizienten wegen der Störfunktion F_{yl}. Als erstes wird die homogene DGL gelöst.

Ansatz für Lösung einer DGL

$$x(t) = e^{\lambda * t} \tag{A.1}$$

Es werden die erste und die zweite Ableitung berechnet und in 3.10 eingesetzt.

Ansatz für die Lösung einer homogene DGL, die Ableitungen der Gleichung A.1

$$\dot{x}(t) = \lambda * e^{\lambda * t} \tag{A.2}$$
$$\ddot{x}(t) = \lambda^2 * e^{\lambda * t} \tag{A.3}$$

Daraus folgt das Homogene DGL-System.

$$m * \lambda^2 * e^{\lambda * t} + \vartheta * \lambda * e^{\lambda * t} = 0 \tag{A.4}$$
$$e^{\lambda * t} * \left(m * \lambda^2 + \vartheta * \lambda\right) = 0 \tag{A.5}$$

Nach umformen bekommt man die folgende Quadratische Gleichung ,des charakteristische Polynom, $m * \lambda^2 + \vartheta * \lambda = 0$.

Lösungen des charakteristischen Polynoms:

$$x_1(t) = -\frac{\lambda * t}{m} \tag{A.6}$$

$$x_2(t) = 1 \tag{A.7}$$

Nach weiterer Rechnung wird daraus die folgende Gleichung.

Die X Richtung:

$$x(t) = \frac{V_x * \cos(\alpha) * m}{\vartheta} * \left(1 - e^{-}\frac{\vartheta * t}{m}\right) \tag{A.8}$$

Analog zur X Richtung wird auch für die Y Richtung die homogene DGL gelöst. Hier ist auch die partielle Lösung der inhomogene DGL dazu zu addieren.

Inhomogene DGL des schiefen Wurfs in Y Richtung:

$$m * \ddot{y} + \vartheta * \dot{y} = -m * g \tag{A.9}$$

In Bartsch (Hans-Jochen, 2001) und in den Skript von Professor Andrerae (Thomas, 2000), kann man den Ansatz für die Störfunktion finden, in diesem Fall bricht der Störfunktion-Ansatz nach der linearen Glied ab, weiter kann man als Ansatz so schreiben:

Ansatz der Störfunktion des DGL:

$$y_p(t) = a_0 + a_1 * t \tag{A.10}$$

$$\dot{y}(t) = a_1 \tag{A.11}$$

$$\ddot{y}(t) = 0 \tag{A.12}$$

Diese eingesetzt in (Formel A.9) ergibt folgende Gleichung:

Lösung der partikulären Inhomogenen DGL:

$$m * 0 + \vartheta * a_1 = -m * g \tag{A.13}$$

$$a_1 = \frac{m * g}{\vartheta} \tag{A.14}$$

$$y_p(t) = a_0 + \frac{m * g * t}{\vartheta} \tag{A.15}$$

Und die gesamte Lösung der inhomogene DGL ist:

$$y(t) = C_1 * e^{-\frac{\vartheta * t}{m}} + C_2 + \left(-\frac{m * g * t}{\vartheta} + a_0\right) \tag{A.16}$$

Nach Einsetzen der Anfangsbedingungen $y(t = 0) = 0$ und $\dot{y}(t = 0) = 0$ werden die Integrationskonstanten C_1 und C_2 gefunden, und man bekommt folgende Lösung:

$$x(t) = \frac{V_x * \cos(\alpha) * m}{\vartheta} * \left(1 - e^{-}\frac{\vartheta * t}{m}\right) \tag{A.17}$$

$$y(t) = \frac{m}{\vartheta^2} * \left((V_y * \sin(\alpha) * \vartheta + m * g) * \left(1 - e^{-}\frac{\vartheta * t}{m}\right) - \vartheta * g * t\right) \tag{A.18}$$

Das ist nicht die endgültige Lösung - man muss weiter logarithmieren weil im Exponential immer noch die veränderliche Variable Zeit(t) steht.

A.4 Verlauf der Wert der Determinante der polynom-regression Matrix

Test auf Singularität der Matrix durch die Determinante. Erst sind die Werte(Ball- und Roboter-Koordinaten) normiert von mm in m umgewandelt. Am Anfang ist der Wert der Determinante sehr nah an NULL bis zum 3, 5 Wert dann Steigt der Wert sehr schnell.

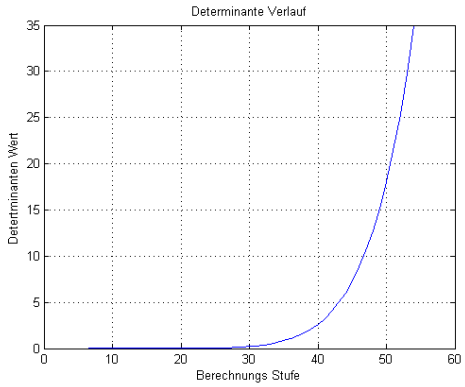


Abbildung A.20: Wert der Determinante verlauf bei normierte Werte

Wenn die Werte der Ball- und Roboter-Koordinaten nicht Normiert(im m von mm Umgewandelt sind) sind dann ist die Determinante nur bei den ersten 2 Werten sehr nah an NULL.

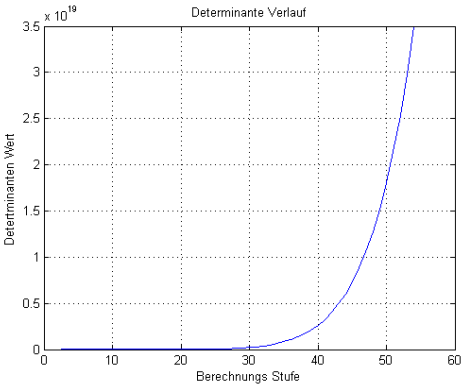


Abbildung A.21: Wert der Determinante verlauf bei nicht normierte Werte

A.5 Vergleich theoretisch berechneten Nullstellen

Tabelle zum Vergleich theoretisch berechneten Nullstellen im Kapitel „Technische Analyse“, mit dem Regression 3.22 implementiert in Matlab.

12	0.454614	1.25611	0.756579
13	0.469547	1.29566	0.758462
14	0.48461	1.33241	0.760157
15	0.50015	1.36533	0.760959
.....			
40	0.868356	1.32207	0.806577
41	0.882743	1.28534	0.808479
42	0.897098	1.24599	0.810338
43	0.911373	1.20406	0.812155
45	0.92566	1.15949	0.814026
46	0.939936	1.11232	0.815797
47	0.954134	1.06257	0.817611
48	0.968175	1.01023	0.819461
49	0.982187	0.955371	0.821222
50	0.996378	0.897985	0.823188
51	1.01042	0.838005	0.825056
52	1.02446	0.775535	0.826913
53	1.03843	0.710489	0.828711
54	1.05244	0.642893	0.830531
55	1.06643	0.57273	0.832242

Abbildung A.22: Tabelle des Ballkoordinaten

A.6 Programm Test, Tabellen

Die zugehörige Tabellen Ausdruck(Robotergeschwindigkeit, Weg, Zeit) zu dem Programmtest 5.2 im Kapitel „Test und Bewertung“.

Fahrzeit	Fahrweg	Winkel	Speed X	Speed Z	Systemtime
802	0.802917	0.529806	862905	-505366	937
460	0.460021	0.821677	680994	-732289	968
1221	1.22147	0.4235	911656	-410954	984
1040	1.04039	0.465046	893801	-448464	0
963	0.963285	0.487972	883286	-468836	15
906	0.906597	0.508537	873458	-486.9	31
828	0.828514	0.539267	858086	-513507	46
742	0.742832	0.580183	836362	-548177	62
676	0.676061	0.618541	814725	-579847	78
629	0.629264	0.649506	796383	-604793	109
582	0.582679	0.68255	775967	-630774	125
544	0.544331	0.710261	758192	-652032	140
474	0.474197	0.760642	724393	-689387	171
443	0.443097	0.788014	705255	-708954	187
414	0.414262	0.815991	685147	-728405	203
389	0.389248	0.837374	669416	-742888	218
365	0.36548	0.85564	655735	-754991	234
343	0.343797	0.870993	644067	-764969	250
322	0.322948	0.887201	631584	-775307	265
306	0.306016	0.899946	621653	-783293	281
290	0.290803	0.910652	613231	-789904	296
277	0.277907	0.915555	609351	-792901	312
265	0.265829	0.921743	604432	-796657	328
254	0.254678	0.926389	600725	-799456	343
241	0.241656	0.936784	592382	-805657	359
228	0.228365	0.946307	584683	-811262	390
214	0.214328	0.962784	571237	-820785	406
199	0.199614	0.977184	559359	-828925	421
184	0.184428	0.987098	551114	-834.43	437
169	0.169479	0.992739	546398	-837526	453
154	0.154497	0.994427	544984	-838447	468
140	0.140297	1.00026	540085	-841611	484
126	0.126271	1.00349	537.36	-843353	500
112	0.112039	1.02037	523053	-852.3	515
97	0.0975427	1.05028	497328	-867562	531
82	0.0825778	1.10572	448.49	-893788	546
67	0.0677663	1.21024	352795	-935701	562
53	0.0539241	1.39794	172001	-985097	578
42	0.0426723	1.70895	-137719	-990471	593
37	0.0372537	01.02.95	-555286	-831659	609
39	0.0395648	2.63606	-874917	-484273	625
46	0.0461016	3.00633	-990867	-134846	640
56	0.056497	-3.0099	-991342	131308	656

Abbildung A.23: Zeit ,Weg, Geschwindigkeit Ausgabe

Die Abweichung der berechneten X und Z Koordinaten des Balls von dem realen.

Koordinate des fliegenden Balls			Berechnete Nullstelle und Z-Koordinate des Balls	
X	Y	Z	Nullstelle	Z-Koordnate
0.517388	0.65949	0.501877	1.#QNAN	-1.#IND
0.538259	0.726893	0.506004	-1.#IND	-1.#IND
0.557895	0.790982	0.509634	-1.#IND	-1.#IND
0.576826	0.852098	0.513126	-1.#IND	-1.#IND
0.595183	0.910046	0.516802	-1.#IND	-1.#IND
0.61282	0.96473	0.520356	-1.#IND	-1.#IND
0.629912	1.01654	0.524136	1.84680	-1.#IND
0.646386	01.01.56	0.527737	-1.#IND	-1.#IND
0.662421	1.11187	0.531281	1.46733	0.762008
0.678773	1.15512	0.534862	2.26630	0.693122
0.694906	1.19572	0.538132	2.08306	0.858221
0.710282	1.23383	0.541414	2.00334	0.822755
0.724671	1.26938	0.544772	1.94490	0.807842
0.740315	1.30264	0.547716	1.86413	0.797616
0.75604	1.33341	0.550741	1.77460	0.781343
0.771227	1.36112	0.55366	1.70405	0.762612
0.786136	1.38647	0.556661	1.65495	0.747541
0.801328	1.40952	0.559243	1.60701	0.736969
0.816919	1.43053	0.561492	1.56958	0.726198
0.832719	1.44865	0.563993	1.53694	0.717026
0.848924	1.46388	0.56637	1.50676	0.700816
0.865384	1.47648	0.568898	1.48064	0.693771
0.88251	1.48645	0.57129	1.45811	0.687399
0.899091	1.49415	0.573964	1.44058	0.682241
0.915336	1.49968	0.577243	1.42731	0.678407
0.933909	1.50098	0.579738	1.41528	0.674676
0.953229	1.49908	0.582099	1.40498	0.671114
0.972113	1.49473	0.584757	1.39706	0.668057
0.991182	1.48707	0.587727	1.39107	0.665521
1.01009	1.47621	0.591131	1.38665	0.663551
1.02861	1.46273	0.594618	1.38355	0.662085
1.04721	1.44664	0.598054	1.38157	0.661015
1.06573	1.42754	0.601675	1.38040	0.660289
1084	1.40528	0.605497	1.37970	0.659858
1.10188	1.37996	0.609347	1.37921	0.659646
1.11948	1.35177	0.613244	1.37883	0.659611
1.13691	1.32087	0.617219	1.37850	0.659728
1.15395	1.28729	0.621227	1.37815	0.659964
1.17058	1.25085	0.625351	1.37768	0.660298
1.18662	1.21175	0.629262	1.37698	0.660668
1.20208	1.17004	0.633015	1.37601	0.661034
1.21736	1.12598	0.637053	1.37485	0.661439
1.23223	1.07959	0.641148	1.37351	0.661876
1.24676	1.03079	0.644959	1.37199	0.662306
1.26135	0.979244	0.648578	1.37037	0.662709
1.27592	0.925241	0.652038	1.36871	0.663079
1.29054	0.869011	0.654911	1.36707	0.663371
1.30424	0.810394	0.658421	1.36533	0.663643
1.31815	0.750491	0.661321	1.36361	0.663864
1.33208	0.68878	0.663737	1.36197	0.664012
1.34615	0.623891	0.666692	1.36041	0.664135

Abbildung A.24: Abweichung zwischen berechneten und realen Koordinaten

A.7 Tabelle zum neuen mathematischem Modell

Die unveränderte Koordinaten des Balls die zum Test für das neue mathematisches Modell 5.4.

Index	X-Koord	Y-Koord	Z-Koord
1	0.849192	0.632368	1.40963
2	0.858973	0.703796	1.39393
3	0.86744	0.775752	1.38058
3	0.875263	0.844937	1.36891
5	0.883045	0.910854	01.01.77
6	0.890712	0.973691	1.34677
7	0.898166	01.01.35	01.01.63
8	0.905472	1.09023	1.32596
9	0.912629	1.14397	1.31573
10	0.919691	01.01.48	1.30566
11	0.926741	1.24278	01.01.58
12	0.933729	1.28815	1.28597
13	0.940781	1.33061	01.01.61
14	0.947838	01.01.05	1.26655
15	0.849192	0.632368	1.40963
...	...	...	...
...	...	...	...
...	...	...	...
40	1.15735	1.45311	1.00622
41	1.16532	1.41919	0.995583
42	1.17301	1.38246	0.985043
43	1.18067	1.34307	0.974658
44	1.18797	01.01.11	0.964364
45	1.19536	1.25701	0.954052
46	1.20257	1.20976	0.944155
47	1.20966	1.15948	0.934612
48	1.21668	01.01.67	0.925103
49	01.01.33	1.05116	0.91585
50	1.23004	0.993265	0.906583
51	1.23678	0.932846	0.897218
52	1.24333	0.870333	0.88796
53	1.24978	0.805324	0.87885
54	1.25607	0.737707	0.870235
55	1.26222	0.667612	0.861991

Abbildung A.25: unveränderte Koordinaten des Balls

A.8 Inhalt der CD-ROM

Diplomarbeit in pdf Format
Quellcode
Implementierte neu Methode zu Ball-Landepunkt Berechnung in Matlab.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit im Sinne der Prüfungsordnung nach §25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Ort, Datum

Unterschrift